Document Title	Specification of Crypto Abstraction Library
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	438
Document Classification	Standard
Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	Changed by	Change Description
4.2.2	AUTOSAR Release Management	• Editorial changes
4.2.1	AUTOSAR Release Management	• Editorial changes
4.1.3	AUTOSAR Release Management	• Missed configuration parameters added • Parameter description of Cpl_<Primitive>_xxx APIs corrected
4.1.2	AUTOSAR Release Management	• Error fixing and consistency improvements • Editorial changes
4.1.1	AUTOSAR Administration	• Services for compression/decompression added • Formal adaptations
4.0.3	AUTOSAR Administration	• CAL0707 and CAL0708_Conf have been removed and the key types structures (e.g. Cal_AsymPrivateKeyType) now explicitly can contain a key handle instead of key data
3.1.5	AUTOSAR Administration	• Integration of key transport services • Key derivation output lenght specified through a parameter • Remove descriptions that reference TRNGs • Complete Configuration parameters
3.1.4	AUTOSAR Administration	• Initial release

Table of Contents

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the software library Crypto Abstraction Library (CAL) to satisfy the top-level requirements represented in the Crypto Requirements Specification (SRS) [CSM_SRS].

The CAL shall provide synchronous services to enable a unique access to basic cryptographic functionalities for all software modules and software components. The functionality required by a software module/component can be different to the functionality required by other software modules/components. For this reason there shall be the possibility to configure the services provided by the CAL individually for all software modules/components.

The construction of the CAL module follows a generic approach. Wherever a detailed specification of structures and interfaces would limit the scope of the usability of the CAL, interfaces and structures are defined in a generic way. This provides an opportunity for future extensions.

2 Acronyms and abbreviations

Acronyms and abbreviations which have a local scope and therefore are not contained in the AUTOSAR glossary [10], are listed in this chapter.

Abbreviation / Acronym:	***Description:***
CAL / Cal	Crypto Abstraction Library
CPL / Cpl	Cryptographic Primitive Library

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[5] Specification of C Implementation Rules
AUTOSAR_TR_CImplementationRules.pdf

[6] Requirement on Libraries
AUTOSAR_SRS_Libraries.pdf

[7] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[8] Requirements on Crypto Service Manager
AUTOSAR_SRS_CryptoServiceManager.pdf

[9] Specification of Crypto Service Manager
AUTOSAR_SWS_CryptoServiceManager.pdf

AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf.pdf

3.2 Related standards and norms

IEC 7498-1 The Basic Model, IEC Norm, 1994

4 Constraints and assumptions

4.1 Limitations

n.a.

4.2 Applicability to car domains

n.a.

5 Dependencies to other modules

[SWS_Cal_00001]
⌈ The CAL shall be able to incorporate cryptographic library modules, which are implemented according to the cryptographic library requirement specification in chapter 8.4. ⌋ ()
[SWS_Cal_00506]
⌈ The CAL shall use the interfaces of the incorporated cryptographic library modules to calculate the result of a cryptographic service.
The incorporated cryptographic library modules provide the implementation of cryptographic routines, e.g. MD5, SHA-1, RSA, AES, Diffie-Hellman key-exchange, etc. ⌋ ()

5.1 File structure

5.1.1 Code file structure

[SWS_Cal_00002]
⌈ The code file structure shall not be defined within this specification completely.
The CAL module shall consist of the following parts: ⌋ ()
[SWS_Cal_00006]
⌈ The code file structure shall contain one or more MISRA-C 2004 conform source files Cal_<xxx>.c, that contain the entire parts of the CAL code. ⌋
(SRS_BSW_00007, SRS_Csm_00036, SRS_BSW_00300)

[SWS_Cal_00534]
⌈ The code file structure shall contain one or more MISRA-C 2004 conform source files Cpl_<xxx>.c, that contain the entire code of the incorporated cryptographic library modules. ⌋ (SRS_BSW_00007, SRS_BSW_00300)

5.1.2 Header file structure

[SWS_Cal_00535]
⌈ The header file structure shall not be defined within this specification completely
The CAL module shall provide the following headers: ⌋ ()

[SWS_Cal_00005]
⌈ The header file structure shall contain an application interface header file Cal.h, that provides the function prototypes to access the CAL services. ⌋
(SRS_LIBS_00005)

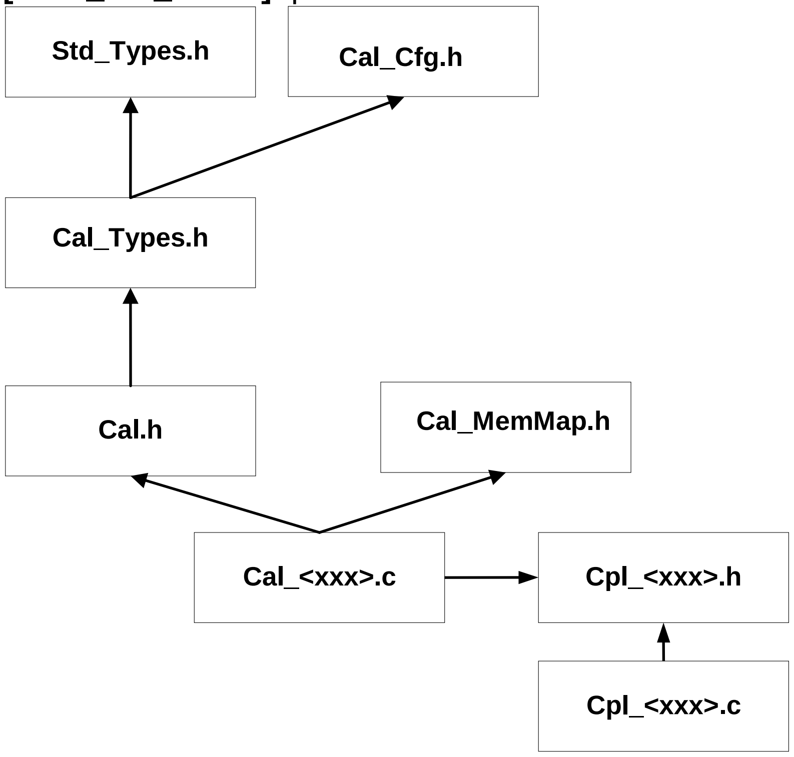
[SWS_Cal_00003]
⌈ The header file structure shall contain a configuration header Cal_Cfg.h, that provides the configuration parameters for the CAL module. ⌋ ()

[SWS_Cal_00004]
⌈ The header file structure shall contain a type header Cal_Types.h, that provides the types, particularly configuration types, for the CAL module. ⌋ ()

[SWS_Cal_00536]
⌈ Each underlying cryptographic library module shall provide a header file Cpl_<xxx>.h. ⌋ ()

[SWS_Cal_00008]
⌈ The Figure in SWS_Cal_00537 (CAL File Structure) shows the include file structure, which shall be as follows:
- Cal.h shall include Cal_Types.h
- Cal_Types.h shall include Cal_Cfg.h
- Cal_Types.h shall include Std_Types.h.
- Cal_<xxx>.c shall include Cal.h and Cal_MemMap.h
- Cal_<xxx>.c shall include Cpl_<xxx>.h
- Cpl_<xxx>.c shall include Cpl_<xxx>.h ⌋ (SRS_BSW_00348)

[SWS_Cal_00537] ⌈



⌋ (SRS_BSW_00301)

6 Requirements traceability

Requirement	Description	Satisfied by
-	-	SWS_Cal_00001
-	-	SWS_Cal_00002
-	-	SWS_Cal_00003
-	-	SWS_Cal_00004
-	-	SWS_Cal_00022
-	-	SWS_Cal_00024
-	-	SWS_Cal_00025
-	-	SWS_Cal_00026
-	-	SWS_Cal_00028
-	-	SWS_Cal_00029
-	-	SWS_Cal_00035
-	-	SWS_Cal_00046
-	-	SWS_Cal_00047
-	-	SWS_Cal_00050
-	-	SWS_Cal_00051
-	-	SWS_Cal_00052
-	-	SWS_Cal_00054
-	-	SWS_Cal_00056
-	-	SWS_Cal_00057
-	-	SWS_Cal_00058
-	-	SWS_Cal_00064
-	-	SWS_Cal_00068
-	-	SWS_Cal_00089
-	-	SWS_Cal_00094
-	-	SWS_Cal_00101
-	-	SWS_Cal_00108
-	-	SWS_Cal_00114
-	-	SWS_Cal_00121
-	-	SWS_Cal_00128
-	-	SWS_Cal_00134
-	-	SWS_Cal_00141
-	-	SWS_Cal_00149
-	-	SWS_Cal_00156
-	-	SWS_Cal_00163
-	-	SWS_Cal_00168
-	-	SWS_Cal_00173

-	-	SWS_Cal_00180
-	-	SWS_Cal_00187
-	-	SWS_Cal_00192
-	-	SWS_Cal_00199
-	-	SWS_Cal_00206
-	-	SWS_Cal_00212
-	-	SWS_Cal_00221
-	-	SWS_Cal_00228
-	-	SWS_Cal_00234
-	-	SWS_Cal_00243
-	-	SWS_Cal_00250
-	-	SWS_Cal_00256
-	-	SWS_Cal_00265
-	-	SWS_Cal_00272
-	-	SWS_Cal_00278
-	-	SWS_Cal_00287
-	-	SWS_Cal_00294
-	-	SWS_Cal_00300
-	-	SWS_Cal_00307
-	-	SWS_Cal_00314
-	-	SWS_Cal_00320
-	-	SWS_Cal_00327
-	-	SWS_Cal_00335
-	-	SWS_Cal_00341
-	-	SWS_Cal_00348
-	-	SWS_Cal_00355
-	-	SWS_Cal_00362
-	-	SWS_Cal_00371
-	-	SWS_Cal_00377
-	-	SWS_Cal_00396
-	-	SWS_Cal_00404
-	-	SWS_Cal_00411
-	-	SWS_Cal_00418
-	-	SWS_Cal_00425
-	-	SWS_Cal_00432
-	-	SWS_Cal_00436
-	-	SWS_Cal_00443
-	-	SWS_Cal_00450
-	-	SWS_Cal_00478

-	-	SWS_Cal_00488
-	-	SWS_Cal_00489
-	-	SWS_Cal_00505
-	-	SWS_Cal_00506
-	-	SWS_Cal_00535
-	-	SWS_Cal_00536
-	-	SWS_Cal_00539
-	-	SWS_Cal_00543
-	-	SWS_Cal_00544
-	-	SWS_Cal_00661
-	-	SWS_Cal_00662
-	-	SWS_Cal_00663
-	-	SWS_Cal_00664
-	-	SWS_Cal_00665
-	-	SWS_Cal_00666
-	-	SWS_Cal_00667
-	-	SWS_Cal_00668
-	-	SWS_Cal_00669
-	-	SWS_Cal_00670
-	-	SWS_Cal_00671
-	-	SWS_Cal_00672
-	-	SWS_Cal_00673
-	-	SWS_Cal_00674
-	-	SWS_Cal_00675
-	-	SWS_Cal_00676
-	-	SWS_Cal_00680
-	-	SWS_Cal_00682
-	-	SWS_Cal_00684
-	-	SWS_Cal_00701
-	-	SWS_Cal_00702
-	-	SWS_Cal_00703
-	-	SWS_Cal_00704
-	-	SWS_Cal_00706
-	-	SWS_Cal_00728
-	-	SWS_Cal_00729
-	-	SWS_Cal_00730
-	-	SWS_Cal_00738
-	-	SWS_Cal_00744
-	-	SWS_Cal_00745

-	-	SWS_Cal_00746
-	-	SWS_Cal_00747
-	-	SWS_Cal_00748
-	-	SWS_Cal_00749
-	-	SWS_Cal_00750
-	-	SWS_Cal_00751
-	-	SWS_Cal_00752
-	-	SWS_Cal_00753
-	-	SWS_Cal_00754
-	-	SWS_Cal_00755
-	-	SWS_Cal_00756
-	-	SWS_Cal_00757
-	-	SWS_Cal_00758
-	-	SWS_Cal_00759
-	-	SWS_Cal_00760
-	-	SWS_Cal_00761
-	-	SWS_Cal_00762
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_Cal_00780
SRS_BSW_00004	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	SWS_Cal_00060
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	SWS_Cal_00006, SWS_Cal_00534, SWS_Cal_00737
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_Cal_00781
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_Cal_00781
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_Cal_00006, SWS_Cal_00534
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_Cal_00537
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	SWS_Cal_00740
SRS_BSW_00305	Data types naming convention	SWS_Cal_00069, SWS_Cal_00073, SWS_Cal_00074, SWS_Cal_00075, SWS_Cal_00079, SWS_Cal_00080, SWS_Cal_00082, SWS_Cal_00086, SWS_Cal_00087, SWS_Cal_00742,

		SWS_Cal_00743
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Cal_00741
SRS_BSW_00307	Global variables naming convention	SWS_Cal_00781
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_Cal_00781
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_Cal_00781
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_Cal_00781
SRS_BSW_00327	Error values naming convention	SWS_Cal_00069
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_Cal_00008, SWS_Cal_00739
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_Cal_00781
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_Cal_00740
SRS_BSW_00402	Each module shall provide version information	SWS_Cal_00780
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Cal_00705
SRS_BSW_00411	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	SWS_Cal_00781
SRS_BSW_00467	The init / deinit services shall only be called by BswM or EcuM	SWS_Cal_00781
SRS_Csm_00001	The CSM shall guarantee that the unused cryptographic primitives of the underlying crypto library are not compiled into the binary	SWS_Cal_00015
SRS_Csm_00004	The CSM shall provide configuration rules and constraints to enable plausibility checks of configuration during ECU configuration time where possible.	SWS_Cal_00030
SRS_Csm_00006	The set of cryptographic services provided by the CSM shall be defined by statical configuration	SWS_Cal_00461
SRS_Csm_00030	The CSM module shall use the	SWS_Cal_00023

	streaming approach for most provided services	
SRS_Csm_00036	The implementation shall be conform to MISRA 2004	SWS_Cal_00006
SRS_LIBS_00002	A library shall be operational before all BSW modules and application SW-Cs	SWS_Cal_00021
SRS_LIBS_00003	A library shall be operational until the shutdown	SWS_Cal_00027
SRS_LIBS_00004	Using libraries shall not pass through a port interface	SWS_Cal_00731
SRS_LIBS_00005	Each library shall provide one header file with its public interface	SWS_Cal_00005
SRS_LIBS_00007	Using a library should be documented	SWS_Cal_00733
SRS_LIBS_00009	All library functions shall be re-entrant	SWS_Cal_00016
SRS_LIBS_00013	The error cases, resulting in the check at runtime of the value of input parameters, shall be listed in SWS	SWS_Cal_00063, SWS_Cal_00067
SRS_LIBS_00015	It shall be possible to configure the microcontroller so that the library code is shared between all callers	SWS_Cal_00734
SRS_LIBS_00018	A library function may only call library functions	SWS_Cal_00736

7 Functional specification

7.1 Basic architecture guidelines

The AUTOSAR library CAL provides other BSW modules and application SWCs with cryptographic services.

The CAL offers C functions that can be called from source code, i.e. from BSW modules, from SWC or from Complex Drivers.

As the CAL is a library, it is not related to a special layer of the AUTOSAR Layered Software Architecture. The services of the CAL are always executed in the context of the calling function.

Many CRY/CPL[1] interfaces use the same cryptographic building blocks. Thus, cryptographic building blocks should be implemented as separate modules and be called from the CRY/CPL interfaces. This implies that the code for cryptographic building blocks should not be implemented more than once.

7.2 General behavior

[SWS_Cal_00016]
⌈ The CAL shall support reentrant access to all services. ⌋ (SRS_LIBS_00009)
[SWS_Cal_00022]
⌈ The CAL shall allow parallel access to different services. ⌋ ()
[SWS_Cal_00035]
⌈ The interface functions shall immediately compute the result, i.e they shall work synchronously. ⌋ ()

7.2.1 Configuration

[SWS_Cal_00025]
⌈ Each service configuration shall be realized as a constant structure of type Cal_<Service>ConfigType . ⌋ ()
[SWS_Cal_00026]
⌈ Each service configuration shall have a name which can be configured. ⌋ ()
[SWS_Cal_00028]
⌈ It shall be possible to create arbitrary many service configurations for each cryptographic service. ⌋ ()
[SWS_Cal_00029]
⌈ When creating a service configuration, it shall be possible to configure all available and allowed schemes and underlying cryptographic primitives. ⌋ ()
[SWS_Cal_00030]

[1] CRY is defined by the Crypto Service Manager (see [8])

⌈ It shall be checked during configuration that only valid service configurations are chosen.⌋ (SRS_Csm_00004)

7.2.2 Normal operation

7.2.2.1 Initialization and shutdown

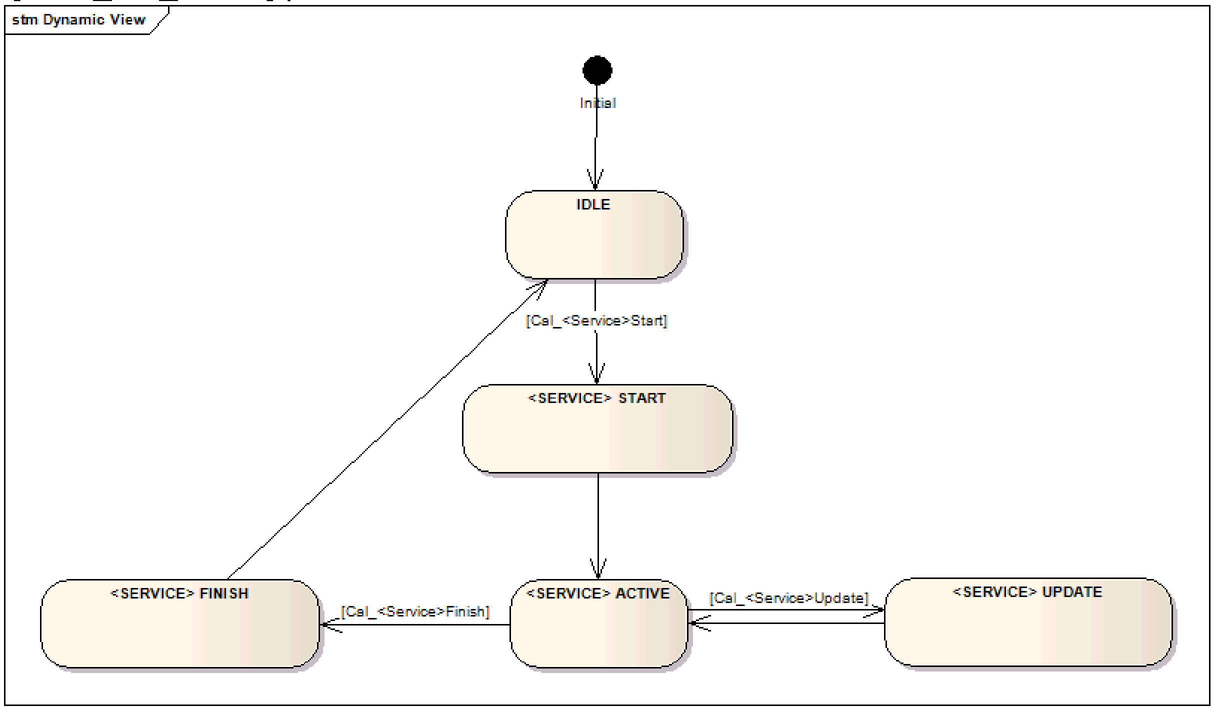
[SWS_Cal_00021]
⌈ The CAL shall not require initialization phase. A Library function may be called at the very first step of ECU initialization, e.g. even by the OS or EcuM, thus the library shall be ready.⌋ (SRS_LIBS_00002)

[SWS_Cal_00027]
⌈ The CAL shall not require a shutdown operation phase.⌋ (SRS_LIBS_00003)

7.2.2.2 Streaming Approach

[SWS_Cal_00023]
⌈ The implementation of those CAL services which expect arbitrary amounts of user data (i.e. the hashing or encryption service) shall be based on the streaming approach with start, update and finish functions. The diagram in SWS_Cal_00024 shows the general design of such a CAL service.⌋ (SRS_Csm_00030)

[SWS_Cal_00024] ⌈



⌋ ()

[SWS_Cal_00728]
⌈ CAL services, which do not expect arbitrary amounts of user data, only have to provide an API Cal_<Service>() (e.g. Cal_RandomGenerate). These services shall be handled as simple function calls. ⌋ ()

[SWS_Cal_00729]
⌈ CAL services, which expect arbitrary amounts of user data, shall provide the APIs Cal_<Service>Start(), Cal_<Service>Update() and Cal_<Service>Finish(). The communication between applications and these CAL services shall follow a strict sequence of steps which is described below. This ensures a reliable communication between applications and the CAL module. ⌋ ()

All applications have to keep with the following rules:

7.2.2.2.1 Initialization

[SWS_Cal_00046]
⌈ The application calls the Cal_<Service>Start request, passing a valid service configuration to the start function. The start function shall check the validity of the configuration it receives. ⌋ ()

[SWS_Cal_00047]
⌈ Cal_<Service>Start shall configure the CAL immediately, set the status of the current service to active, store the status of the service and all necessary context in the context buffer, and return. ⌋ ()

7.2.2.2.2 Update

The application provides the data necessary for the computation of the intended service.

[SWS_Cal_00050]
⌈ The application calls the Cal_<Service>Update request, passing data which is necessary for the computation of the service to the update function. The update function shall check whether the current service is already initialized. ⌋ ()

[SWS_Cal_00051]
⌈ The CAL shall assume that the data provided to Cal_<Service>Update will not change until it returns. ⌋ ()

[SWS_Cal_00052]
⌈ If the service has been initialized before, the update function shall immediately process the given data, set the status of the current service again to active, store the status of the service and all necessary context in the context buffer, and return the status of the update. ⌋ ()

[SWS_Cal_00054]
⌈ The CAL shall allow the application to call the update function arbitrarily often. ⌋ ()

7.2.2.2.3 Finish

The application provides the result buffer necessary for the finishing of the computation of the intended service.

[SWS_Cal_00056]
⌈ The application calls the Cal_<Service>Finish request, passing the result buffer and optional data which is necessary for the finishing of the cryptographic service to the finish function. The finish function shall check whether the current service is already initialized. ⌋ ()

[SWS_Cal_00057]
⌈ The CAL shall assume that the data provided to Cal_<Service>Finish will not change until it returns. ⌋ ()

[SWS_Cal_00058]
⌈ If the service has been initialized before, the finish function shall immediately process the given data, finish the computation of the current cryptographic service, set the status of the service in the context buffer to idle, store the result of the service in the result buffer, and return the status of the finishing. ⌋ ()

7.2.2.3 Context of services

As the CAL is a library, it is not allowed to store any internal states.

When calling a service of the CAL, the application has to provide a pointer to a buffer, in which the CAL can store all context and status information that is necessary to process the service. This context buffer has to be provided consistently to all calls of the Start-, Update- and Finish-APIs belonging to one service request cycle.

[SWS_Cal_00730]

⌈ The size of the context buffer, that has to be provided by the caller, depends on the selected service and on the selected CPL method.
The CAL part of the configuration tool shall generate a macro that contains the desired size of the context buffer for each service configuration. ⌋ ()

All context buffers shall be aligned according to the maximum alignment of all scalar types on the given platform.

7.3 Version check

[SWS_Cal_00060]
⌈ The CAL module shall perform Inter Module Checks to avoid integration of incompatible files.
The imported included files shall be checked by preprocessing directives. ⌋ (SRS_BSW_00004)

The following version numbers shall be verified:
< MAB >_AR_RELEASE_MAJOR_VERSION
< MAB >_AR_RELEASE_MINOR_VERSION
where <MAB> is the module module abbreviation of the other (external) modules which provide header files included by the CAL module.

If the values are not identical to the expected values, an error shall be reported.

7.4 Error detection

[SWS_Cal_00063]
⌈ Functions of the CAL should check at runtime (both in production and development code) the value of input parameters, especially cases where erroneous value can bring to fatal error or unpredictable result, if they have the values allowed by the function specification. All the error cases shall be listed in SWS and the function should return a specified value (in SWS) that is not configurable. This value is dependant of the function and the error case so it is determined case by case. ⌋ (SRS_LIBS_00013)

[SWS_Cal_00064]
⌈ The API parameters shall be checked in the order in which they are passed. ⌋ ()

[SWS_Cal_00488]
⌈ If an error is detected, the desired service shall return with CAL_E_NOT_OK. ⌋ ()

[SWS_Cal_00489]

⌈ The following table specifies which errors shall be evaluated for each API call: ⌋ ()

[SWS_Cal_00539] ⌈

API call	*Error condition*	*API return value*
All APIs that have a pointer as parameter	Pointer is Nullpointer	All APIs shall return CAL_E_NOT_OK or void resp.
Cal_<Service>Update	Service is not initialized	CAL_E_NOT_OK
Cal_<Service>Finish	Service is not initialized	CAL_E_NOT_OK
Cal_<Service>Start	Invalid cryptographic method for selected service	CAL_E_NOT_OK
Cal_<Service>	Invalid cryptographic method for selected service	CAL_E_NOT_OK
Cal_MacGenerateStart	Invalid key type for selected service	CAL_E_NOT_OK
Cal_MacVerifyStart		
Cal_SymBlockEncryptStart		
Cal_SymBlockDecryptStart		
Cal_SymEncryptStart		
Cal_SymDecryptStart		
Cal_AsymEncryptStart		
Cal_AsymDecryptStart		
Cal_KeyExchangeCalcPubVal		
Cal_KeyExchangeCalcSecretStart		
Cal_SymKeyWrapSymStart		
Cal_SymKeyWrapAsymStart		
Cal_AsymPrivateKeyWrapSymStart		
Cal_AsymPrivateKeyWrapAsymStart		
Cal_AsymPublicKeyExtractStart		
Cal_SignatureGenerateStart		
Cal_SignatureVerifyStart		

⌋ ()

7.5 Error notification

[SWS_Cal_00067]
⌈ The functions of the CAL shall not call the DET in case of error. ⌋ (SRS_LIBS_00013)

7.6 Using Library API

[SWS_Cal_00731]

⌈ CAL API can be directly called from BSW modules or SWC. No port definition is required. It is a pure function call. ⌋ (SRS_LIBS_00004)

The statement `#include "Cal.h"` shall be placed by the developer or an application code generator but not by the RTE generator

[SWS_Cal_00733]
⌈ Using a library shall be documented. If a BSW module or a SWC uses a Library, the developer shall add an Implementation-DependencyOnLibrary in the BSW/SWC template.
minVersion and maxVersion parameters correspond to the supplier version. In case of AUTOSAR library, these parameters may be left empty because a SWC or BSW module may rely on a library behaviour, not on a supplier implementation. However, the SWC or BSW modules shall be compatible with the AUTOSAR platform where they are integrated. ⌋ (SRS_LIBS_00007)

7.7 Library implementation

[SWS_Cal_00015]
⌈ Due to memory restrictions the CAL Library and the underlying Crypto Library shall only provide those services and algorithms which are necessary for the applications running on the ECU. Therefore parts of the CAL Library have to be generated based on a configuration that describes which cryptographic methods are necessary for the applications. ⌋ (SRS_Csm_00001)

[SWS_Cal_00734]
⌈ The CAL shall be implemented in a way that the code can be shared among callers in different memory partitions. ⌋ (SRS_LIBS_00015)
[SWS_Cal_00736]
⌈ A library function shall not call any BSW modules functions. A library function can call other library functions. Because a library function shall be reentrant. But other BSW modules functions may not be reentrant. ⌋ (SRS_LIBS_00018)

[SWS_Cal_00737]
⌈ The library, written in C programming language, should conform to the HIS subset of the MISRA C Standard.
Only in technically reasonable, exceptional cases MISRA violations are permissible. Such violations against MISRA rules shall be clearly identified and documented within comments in the C source code (including rationale why MISRA rule is violated). The comment shall be placed right above the line of code which causes the violation and have the following syntax:
/* MISRA RULE XX VIOLATION: This the reason why the MISRA rule could not be followed in this special case*/ ⌋ (SRS_BSW_00007)

[SWS_Cal_00738]
⌈ Each AUTOSAR library Module implementation <library>*.c shall include the header file MemMap.h. ⌋ ()

[SWS_Cal_00739]
⌈ Each AUTOSAR library Module implementation <library>*.c, that uses AUTOSAR integer data types and/or the standard return, shall include the header file Std_Types.h. ⌋ (SRS_BSW_00348)

[SWS_Cal_00740]
⌈ All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of native C data types, unless this library is clearly identified to be compliant only with a platform. ⌋ (SRS_BSW_00304, SRS_BSW_00378)

[SWS_Cal_00741]
⌈ All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, unless this library is clearly identified to be compliant only with a platform. ⌋ (SRS_BSW_00306)

8 API specification

8.1 Imported types

[SWS_Cal_00068]
⌈ Only the standard AUTOSAR types provided by Std_Types.h shall be imported. ⌋
()

8.2 Type definitions

8.2.1 API types

8.2.1.1 Cal_ReturnType

[SWS_Cal_00069] ⌈

Name:	`Cal_ReturnType`	
Type:	`Enumeration`	
Range:	`CAL_E_OK`	The execution of the called function succeeded / the result of the called function is "ok". This return code shall be given as value "0"
	`CAL_E_NOT_OK`	The execution of the called function failed / the result of the called function is "not ok". This return code shall be given as value "1".
	`CAL_E_SMALL_BUFFER`	The service request failed because the provided buffer is too small to store the result of the service. This return code shall be given as value "3".
	`CAL_E_ENTROPY_EXHAUSTION`	The service request failed because the entropy of the random number generator is exhausted. This return code shall be given as value "4".
Description:	Enumeration of the return type of the CAL module	

⌋ (SRS_BSW_00305, *SRS_BSW_00327*)

8.2.1.2 Cal_ConfigIdType

[SWS_Cal_00073] ⌈

Name:	`Cal_ConfigIdType`
Type:	`uint16`
Description:	Identification of a CAL service configuration via a numeric identifier that is unique within a service. The name of a CAL service configuration, i.e. the name of the container Cal_<Service>Config, shall serve as a symbolic name for this parameter. Range: 0..65535

⌋ (SRS_BSW_00305)

8.2.1.3 Cal_<Service>ConfigType

[SWS_Cal_00074] ⌈

Name:	Cal_<Service>ConfigType		
Type:	Structure		
Element:	Cal_ConfigIdType	ConfigId	The numeric identifier of a configuration.
	Cal_ReturnType	(*PrimitiveStartFct)(<primitive parameter list>)	This element shall only exist if the service contains the function Cal_<Service>Start. It is a pointer to the function Cpl_<Primitive>Start of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cpl_<Primitive>Start.
	Cal_ReturnType	(*PrimitiveUpdateFct)(<primitive parameter list>)	This element shall only exist if the service contains the function Cal_<Service>Update. It is a pointer to the function Cpl_<Primitive>Update of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cpl_<Primitive>Update.
	Cal_ReturnType	(*PrimitiveFinishFct)(<primitive parameter list>)	This element shall only exist if the service contains the function Cal_<Service>Finish. It is a pointer to the function Cpl_<Primitive>Finish of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cpl_<Primitive>Finish.
	Cal_ReturnType	(*PrimitiveFct)(<primitive parameter list>)	This element shall only exist if the service contains the function Cal_<Service>. It is a pointer to the function Cpl_<Primitive> of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cpl_<Primitive>.
	void	*PrimitiveConfigPtr	A pointer to the configuration of the underlying cryptographic primitive
Description:	Data structure which shall encompass all information needed to specify the cryptographic primitives needed for the <Service> cryptographic service. It shall furthermore contain information on the callback function.		

⌋ (SRS_BSW_00305)

8.2.1.4 Cal_AlignType

[SWS_Cal_00743] ⌈

Name:	Cal_AlignType
Type:	<maxAlignScalarType>
Description:	A scalar type which has maximum alignment restrictions on the given platform.

	This value is configured by "CalMaxAlignScalarType". <maxAlignScalarType> can be e.g. uint8, uint16 or uint32. All context buffers shall be aligned according to the maximum alignment of all scalar types on the given platform.

⌋ (SRS_BSW_00305)

8.2.1.5 Cal_<Service>CtxBufType

[SWS_Cal_00742] ⌈

Name:	`Cal_<Service>CtxBufType`
Type:	`Cal_AlignType[CAL_<SERVICE>_CONTEXT_BUFFER_SIZE]`
Description:	Type definition of the context buffer of a service. CAL_<SERVICE>_CONTEXT_BUFFER_SIZE shall be chosen such that "CAL_<SERVICE>_CONTEXT_BUFFER_SIZE * sizeof(Cal_AlignType)" is greater or equal "Cal<Service>MaxCtxBufferByteSize".

⌋ (SRS_BSW_00305)

8.2.1.6 Cal_VerifyResultType

[SWS_Cal_00075]⌈

Name:	`Cal_VerifyResultType`	
Type:	`Enumeration`	
Range:	`CAL_E_VER_OK`	The result of the verification is "true", i.e. the two compared elements are identical. This return code shall be given as value "0"
	`CAL_E_VER_NOT_OK`	The result of the verification is "false", i.e. the two compared elements are not identical. This return code shall be given as value "1".
Description:	Enumeration of the result type of verification operations.	

⌋ (SRS_BSW_00305)

8.2.1.7 Cal_AsymPublicKeyType

[SWS_Cal_00079] ⌈

Name:	`Cal_AsymPublicKeyType`		
Type:	`Structure`		
Element:	`uint32`	`length`	This element contains the length of the key stored in element 'data'
	`Cal_AlignType [CAL_ASYM_PUB_KEY_MAX_SIZE]`	`data`	This element contains the key data or a key handle.
Description:	Structure for the public asymmetrical key. CAL_ASYM_PUB_KEY_MAX_SIZE shall be chosen such that "CAL_ASYM_PUB_KEY_MAX_SIZE * sizeof(Cal_AlignType)" is greater or equal to the maximum of the configured values CalAsymEncryptMaxKeySize, CalSignatureVerifyMaxKeySize, CalAsymPublicKeyExtractMaxKeySize, CalSymKeyWrapAsymMaxPubKeySize and CalAsymPrivateKeyWrapAsymMaxPubKeySize.		

⌋ (SRS_BSW_00305)

8.2.1.8 Cal_AsymPrivateKeyType

[SWS_Cal_00080] ⌈

Name:	`Cal_AsymPrivateKeyType`		
Type:	`Structure`		
Element:	`uint32`	`length`	This element contains the length of the key stored in element 'data'
	`Cal_AlignType[CAL_ASYM_PRIV_KEY_MAX_SIZE]`	`data`	This element contains the key data or a key handle.
Description:	Structure for the private asymmetrical key. CAL_ASYM_PRIV_KEY_MAX_SIZE shall be chosen such that "CAL_ASYM_PRIV_KEY_MAX_SIZE * sizeof(Cal_AlignType)" is greater or equal to the maximum of the configured values CalAsymDecryptMaxKeySize, CalSignatureGenerateMaxKeySize, CalAsymPrivateKeyExtractMaxKeySize, CalAsymPrivateKeyWrapSymMaxPrivKeySize and CalAsymPrivateKeyWrapAsymMaxPrivKeySize.		

⌋ (SRS_BSW_00305)

8.2.1.9 Cal_SymKeyType

[SWS_Cal_00082]⌈

Name:	`Cal_SymKeyType`		
Type:	`Structure`		
Element:	`uint32`	`length`	This element contains the length of the key stored in element 'data'
	`Cal_AlignType[CAL_SYM_KEY_MAX_SIZE]`	`data`	This element contains the key data or a key handle.
Description:	Structure for the symmetrical key. CAL_SYM_KEY_MAX_SIZE shall be chosen such that "CAL_SYM_KEY_MAX_SIZE * sizeof(Cal_AlignType)" is greater or equal to the maximum of the configured values CalSymBlockEncryptMaxKeySize, CalSymBlockDecryptMaxKeySize, CalSymEncryptMaxKeySize, CalSymDecryptMaxKeySize, CalKeyDeriveMaxKeySize, CalSymKeyExtractMaxKeySize, CalMacGenerateMaxKeySize, CalMacVerifyMaxKeySize, CalSymKeyWrapSymMaxSymKeySize, CalSymKeyWrapAsymMaxSymKeySize and CalAsymPrivateKeyWrapSymMaxSymKeySize.		

⌋ (SRS_BSW_00305)

8.2.1.10 Cal_KeyExchangeBaseType

[SWS_Cal_00086] ⌈

Name:	`Cal_KeyExchangeBaseType`		
Type:	`Structure`		
Element:	`uint32`	`length`	This element contains the length of the key stored in element 'data'
	`Cal_AlignType [CAL_KEY_EX_BASE_MAX_SIZE]`	`data`	This element contains the key data or a key handle.
Description:	Structure with base type information of the key exchange protocol. CAL_KEY_EX_BASE_MAX_SIZE shall be chosen such that		

	"CAL_KEY_EX_BASE_MAX_SIZE * sizeof(Cal_AlignType)" is greater or equal to the maximum of the configured values CalKeyExchangeCalcPubValMaxBaseTypeSize and CalKeyExchangeCalcSecretMaxBaseTypeSize

⌋ (SRS_BSW_00305)

8.2.1.11 Cal_KeyExchangePrivateType

[SWS_Cal_00087] ⌈

Name:	`Cal_KeyExchangePrivateType`		
Type:	`Structure`		
Element:	`uint32`	`length`	This element contains the length of the key stored in element 'data'
	`Cal_AlignType[CAL_KEY_EX_PRIV_MAX_SIZE]`	`data`	This element contains the key data or a key handle.
Description:	Structure with the private Information of the key exchange protocol only known to the current user. CAL_KEY_EX_PRIV_MAX_SIZE shall be chosen such that "CAL_KEY_EX_PRIV_MAX_SIZE * sizeof(Cal_AlignType)" is greater or equal to the maximum of the configured values CalKeyExchangeCalcPubValMaxPrivateTypeSize and CalKeyExchangeCalcSecretMaxPrivateTypeSize		

⌋ (SRS_BSW_00305)

8.3 API functions

[SWS_Cal_00478]
⌈ As the CAL is a library, all functions have to be reentrant. ⌋ ()

8.3.1 General interfaces

8.3.1.1 Cal_GetVersionInfo

[SWS_Cal_00705]⌈

Service name:	Cal_GetVersionInfo	

Syntax:	`void Cal_GetVersionInfo(`
	`    Std_VersionInfoType* versioninfo`
	`)`	

Service ID[hex]:	0x3B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.

Return value:	void	none
Description:	Returns the version information of this module.	

⌋ (SRS_BSW_00407)

[SWS_Cal_00706]
⌈ The function Cal_GetVersionInfo shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (SRS_BSW_00407). ⌋ ()

[SWS_Cal_00762]
⌈ If the provided 'versioninfo' is a NULL pointer, Cal_GetVersionInfo shall return immediately without any further action and especially not write at NULL. ⌋ ()

8.3.2 Hash interface

A cryptographic hash function is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the hash value, such that an accidental or intentional change to the data will change the hash value. Main properties of hash functions are that it is infeasible to find a message that has a given hash or to find two different messages with the same hash.

8.3.2.1 Cal_HashStart

[SWS_Cal_00089] ⌈

Service name:	Cal_HashStart	

Syntax:	`Cal_ReturnType Cal_HashStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_HashCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration that has to be used during the hash value computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the hash service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_HashStart.

8.3.2.2 Cal_HashUpdate

[SWS_Cal_00094] ⌈

Service name:	Cal_HashUpdate	

Syntax:	`Cal_ReturnType Cal_HashUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_HashCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration that has to be used during the hash value computation.
	dataPtr	Holds a pointer to the data to be hashed
	dataLength	Contains the number of bytes to be hashed.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the hash service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The hash computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_HashUpdate.

8.3.2.3 Cal_HashFinish

[SWS_Cal_00101] ⌈

Service name:	Cal_HashFinish	

Syntax:	`Cal_ReturnType Cal_HashFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_HashCtxBufType contextBuffer,`
	`    uint8* resultPtr,`
	`    uint32* resultLengthPtr,`
	`    boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration that has to be used during the hash value computation.

	TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: Truncation is allowed. FALSE: Truncation is not allowed.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	resultLengthPtr	Holds a pointer to the 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. On returning from this function the actual length of the computed value shall be stored.
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the hash value computation. If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result, and truncation was not allowed.
Description:	This function shall be used to finish the hash service of the CAL module. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The hash computation is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00661]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small and truncation is allowed, the result of the computation shall be truncated to the size of the provided buffer, and `CAL_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_HashFinish.

8.3.3 MAC interface

A message authentication code (MAC) is a short piece of information used to authenticate a message. A MAC algorithm accepts as input a secret key and an arbitrary-length message to be authenticated, and outputs a MAC. The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content.

8.3.3.1 Cal_MacGenerateStart

[SWS_Cal_00108] ⌈

Service name:	Cal_MacGenerateStart	

Syntax:	`Cal_ReturnType Cal_MacGenerateStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_MacGenerateCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC computation.
	keyPtr	Holds a pointer to the key necessary for the MAC generation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the MAC generate service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_MacGenerateStart.

8.3.3.2 Cal_MacGenerateUpdate

[SWS_Cal_00114] ⌈

Service name:	Cal_MacGenerateUpdate	

Syntax:	`Cal_ReturnType Cal_MacGenerateUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_MacGenerateCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC computation.
	dataPtr	Holds a pointer to the data for which a MAC shall be computed.
	dataLength	Contains the number of bytes for which the MAC shall be computed.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	

Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the MAC generate service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The MAC computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_MacGenerateUpdate.

8.3.3.3 Cal_MacGenerateFinish

[SWS_Cal_00121] ⌈

Service name:	Cal_MacGenerateFinish	

Syntax:	`Cal_ReturnType Cal_MacGenerateFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_MacGenerateCtxBufType contextBuffer,`
	`    uint8* resultPtr,`
	`    uint32* resultLengthPtr,`
	`    boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC computation.
	TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: Truncation is allowed. FALSE: Truncation is not allowed.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	resultLengthPtr	Holds a pointer to the 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. On returning from this function the actual length of the computed MAC shall be stored.
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the MAC generation. If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result, and truncation was not allowed.
Description:	This function shall be used to finish the MAC generation service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK".	

	Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The MAC computation is done by the underlying primitive.

⌋ ()

[SWS_Cal_00662]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small and truncation is allowed, the result of the computation shall be truncated to the size of the provided buffer, and `CAL_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_MacGenerateFinish.

8.3.3.4 Cal_MacVerifyStart

[SWS_Cal_00128] ⌈

Service name:	Cal_MacVerifyStart	

Syntax:	`Cal_ReturnType Cal_MacVerifyStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_MacVerifyCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC verification.
	keyPtr	Holds a pointer to the key necessary for the MAC verification.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the MAC verify service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_MacVerifyStart.

8.3.3.5 Cal_MacVerifyUpdate

[SWS_Cal_00134] ⌈

Service name:	Cal_MacVerifyUpdate

Syntax:	`Cal_ReturnType Cal_MacVerifyUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_MacVerifyCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x0A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC verification.
	dataPtr	Holds a pointer to the data for which a MAC shall be verified.
	dataLength	Contains the number of bytes for which the MAC shall be verified.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the MAC verification service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The MAC computation is done by the underlying primitive.The MAC computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_MacVerifyUpdate.

8.3.3.6 Cal_MacVerifyFinish

[SWS_Cal_00141] ⌈

Service name:	Cal_MacVerifyFinish	

Syntax:	`Cal_ReturnType Cal_MacVerifyFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_MacVerifyCtxBufType contextBuffer,`
	`    const uint8* MacPtr,`
	`    uint32 MacLength,`
	`    Cal_VerifyResultType* resultPtr`
	`)`	

Service ID[hex]:	0x0B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC verification.
	MacPtr	Holds a pointer to the memory location which will hold the MAC to verify.
	MacLength	Holds the length of the MAC to be verified.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the

		result of the MAC verification.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the MAC verification service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The MAC computation is done by the underlying primitive.The MAC computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_MacVerifyFinish.

8.3.4 Random interface

The random interface provides generation of random numbers. The randomness of pseudo random number generators can be increased by an appropriate selection of the seed.

8.3.4.1 Cal_RandomSeedStart

[SWS_Cal_00149] ⌈

Service name:	Cal_RandomSeedStart	

Syntax:	`Cal_ReturnType Cal_RandomSeedStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_RandomCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x0C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the seeding of the random number generator.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the random seed service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_RandomSeedStart.

8.3.4.2 Cal_RandomSeedUpdate

[SWS_Cal_00156] ⌈

Service name:	Cal_RandomSeedUpdate	

Syntax:	`Cal_ReturnType Cal_RandomSeedUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_RandomCtxBufType contextBuffer,`
	`    const uint8* seedPtr,`
	`    uint32 seedLength`
	`)`	

Service ID[hex]:	0x0D	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the seeding of the random number generator.
	seedPtr	Holds a pointer to the seed for the random number generator.
	seedLength	Contains the length of the seed in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed a seed to the random number generator. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The seeding of the random number generator is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_RandomSeedUpdate.

8.3.4.3 Cal_RandomSeedFinish

[SWS_Cal_00163] ⌈

Service name:	Cal_RandomSeedFinish	

Syntax:	`Cal_ReturnType Cal_RandomSeedFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_RandomCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x0E	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the seeding of the random number generator.
Parameters	contextBuffer	Holds the pointer to the buffer in which the context of this

(inout):		service can be stored
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the random seed service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The seeding of the random number generator is done by the underlying primitive	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_RandomSeedFinish.

8.3.4.4 Cal_RandomGenerate

[SWS_Cal_00543] ⌈

Service name:	Cal_RandomGenerate	

Syntax:	`Cal_ReturnType Cal_RandomGenerate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_RandomCtxBufType contextBuffer,`
	`    uint8* resultPtr,`
	`    uint32 resultLength`
	`)`	

Service ID[hex]:	0x0F	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during random number generation
	resultLength	Holds the amount of random bytes which should be generated.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored. If a seed is needed, this must be the same context buffer that has been used for the call of the RandomSeed interfaces.
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the random number generation. The memory location must have at least the size "resultLength".
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_ENTROPY_EXHAUSTION: Request failed, entropy of random number generator is exhausted.
Description:	This function shall be used to start the random number generation service of the CAL module. The function shall call the function Cpl_<Primitive> of the primitive which is identified by the "cfgId" and return the value returned by that function.	

⌋ ()

The generation of a random number is based on the seed, which was previously set with the interfaces Cal_RandomSeedStart, Cal_RandomSeedUpdate, and

Cal_RandomSeedFinish. These interfaces follow the streaming approach. Thus it is possible to feed the seed e.g. from different sources.

To generate a random number, no streaming approach is necessary. The interface Cal_RandomGenerate can be called arbitrarily often to generate multiple random numbers.

The APIs of the Random service are designed for usage of pseudo random number generators (PRNGs). True random number generators (TRNGs) are not supported.

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_RandomGenerate.

8.3.5 Symmetrical block interface

A block cipher is a symmetric key cipher operating on fixed-length blocks, with an unvarying transformation. A block cipher encryption algorithm might take (for example) a 128-bit block of plaintext as input, and output a corresponding 128-bit block of ciphertext. The exact transformation is controlled using a second input — the secret key. Decryption is similar: the decryption algorithm takes, in this example, a 128-bit block of ciphertext together with the secret key, and yields the original 128-bit block of plaintext.

8.3.5.1 Cal_SymBlockEncryptStart

[SWS_Cal_00168] ⌈

Service name:	Cal_SymBlockEncryptStart	

Syntax:	`Cal_ReturnType Cal_SymBlockEncryptStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymBlockEncryptCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical block encryption computation.
	keyPtr	Holds a pointer to the key which has to be used during the symmetrical block encryption computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the symmetrical block encrypt service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store	

	this state in the context buffer.

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymBlockEncryptStart.

8.3.5.2 Cal_SymBlockEncryptUpdate

[SWS_Cal_00173] ⌈

Service name:	Cal_SymBlockEncryptUpdate	

Syntax:	`Cal_ReturnType Cal_SymBlockEncryptUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymBlockEncryptCtxBufType contextBuffer,`
	`    const uint8* plainTextPtr,`
	`    uint32 plainTextLength,`
	`    uint8* cipherTextPtr,`
	`    uint32* cipherTextLengthPtr`
	`)`	

Service ID[hex]:	0x11	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical block encryption computation.
	plainTextPtr	Holds a pointer to the plain text that shall be encrypted.
	plainTextLength	Contains the length of the plain text in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	cipherTextLengthPtr	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 cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
Parameters (out):	cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to feed the symmetrical block encryption service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The encryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00663]

⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymBlockEncryptUpdate.

8.3.5.3 Cal_SymBlockEncryptFinish

[SWS_Cal_00180] ⌈

Service name:	Cal_SymBlockEncryptFinish	

Syntax:	`Cal_ReturnType Cal_SymBlockEncryptFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymBlockEncryptCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical block encryption computation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the symmetrical block encryption service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The encryption process is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymBlockEncryptFinish.

8.3.5.4 Cal_SymBlockDecryptStart

[SWS_Cal_00187] ⌈

Service name:	Cal_SymBlockDecryptStart	

Syntax:	`Cal_ReturnType Cal_SymBlockDecryptStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymBlockDecryptCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the constant CAL module configuration

		which has to be used during the symmetrical block decryption computation.
	keyPtr	Holds a pointer to the key which has to be used during the symmetrical block decryption computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the symmetrical block decrypt service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymBlockDecryptStart.

8.3.5.5 Cal_SymBlockDecryptUpdate

[SWS_Cal_00192] ⌈

Service name:	Cal_SymBlockDecryptUpdate	

Syntax:	`Cal_ReturnType Cal_SymBlockDecryptUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymBlockDecryptCtxBufType contextBuffer,`
	`    const uint8* cipherTextPtr,`
	`    uint32 cipherTextLength,`
	`    uint8* plainTextPtr,`
	`    uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical block decryption computation.
	cipherTextPtr	Holds a pointer to the constant cipher text that shall be decrypted.
	cipherTextLength	Contains the length of the cipher text in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	plainTextLengthPtr	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 plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
Parameters (out):	plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed

		CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to feed the symmetrical block decryption service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The decryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00664]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned.
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymBlockDecryptUpdate. ⌋ ()

8.3.5.6 Cal_SymBlockDecryptFinish

[SWS_Cal_00199] ⌈

Service name:	Cal_SymBlockDecryptFinish	

Syntax:	`Cal_ReturnType Cal_SymBlockDecryptFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymBlockDecryptCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical block decryption computation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the symmetrical block decryption service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The decryption process is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymBlockDecryptFinish.

8.3.6 Symmetrical interface

Symmetric-key algorithms are algorithms that use identical cryptographic keys for both decryption and encryption. The keys, in practice, represent a shared secret between two or more parties.

8.3.6.1 Cal_SymEncryptStart

[SWS_Cal_00206] ⌈

Service name:	Cal_SymEncryptStart	

Syntax:	`Cal_ReturnType Cal_SymEncryptStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymEncryptCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr,`
	`    const uint8* InitVectorPtr,`
	`    uint32 InitVectorLength`
	`)`	

Service ID[hex]:	0x16	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical encryption computation.
	keyPtr	Holds a pointer to the key which has to be used during the symmetrical encryption computation.
	InitVectorPtr	Holds a pointer to the initialisation vector which has to be used during the symmetrical encryption computation.
	InitVectorLength	Holds the length of the initialisation vector which has to be used during the symmetrical encryption computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the symmetrical encrypt service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymEncryptStart.

8.3.6.2 Cal_SymEncryptUpdate

[SWS_Cal_00212] ⌈

Service name:	Cal_SymEncryptUpdate

Syntax:	`Cal_ReturnType Cal_SymEncryptUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymEncryptCtxBufType contextBuffer,`

	`const uint8* plainTextPtr,`
	`uint32 plainTextLength,`
	`uint8* cipherTextPtr,`
	`uint32* cipherTextLengthPtr`
	`)`	

Service ID[hex]:	0x17	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical encryption computation.
	plainTextPtr	Holds a pointer to the plain text that shall be encrypted.
	plainTextLength	Contains the length of the plain text in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	cipherTextLengthPtr	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 cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
Parameters (out):	cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to feed the symmetrical encryption service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The encryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00665]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymEncryptUpdate.

8.3.6.3 Cal_SymEncryptFinish

[SWS_Cal_00221]⌈

Service name:	Cal_SymEncryptFinish

Syntax:	`Cal_ReturnType Cal_SymEncryptFinish(`
	`Cal_ConfigIdType cfgId,`
	`Cal_SymEncryptCtxBufType contextBuffer,`
	`uint8* cipherTextPtr,`
	`uint32* cipherTextLengthPtr`

	)	
Service ID[hex]:	0x18	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical encryption computation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	cipherTextLengthPtr	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 cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
Parameters (out):	cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to finish the symmetrical encryption service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The encryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00666]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymEncryptFinish.

8.3.6.4 Cal_SymDecryptStart

[SWS_Cal_00228] ⌈

Service name:	Cal_SymDecryptStart	

Syntax:	`Cal_ReturnType Cal_SymDecryptStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymDecryptCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr,`
	`    const uint8* InitVectorPtr,`
	`    uint32 InitVectorLength`
	`)`	

Service ID[hex]:	0x19	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical decryption

		computation.
	keyPtr	Holds a pointer to the key which has to be used during the symmetrical decryption computation.
	InitVectorPtr	Holds a pointer to the initialisation vector which has to be used during the symmetrical decryption computation.
	InitVectorLength	Holds the length of the initialisation vector which has to be used during the symmetrical decryption computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the symmetrical decrypt service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymDecryptStart.

8.3.6.5 Cal_SymDecryptUpdate

[SWS_Cal_00234] ⌈

Service name:	Cal_SymDecryptUpdate	

Syntax:	`Cal_ReturnType Cal_SymDecryptUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymDecryptCtxBufType contextBuffer,`
	`    const uint8* cipherTextPtr,`
	`    uint32 cipherTextLength,`
	`    uint8* plainTextPtr,`
	`    uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x1A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical decryption computation.
	cipherTextPtr	Holds a pointer to the constant cipher text that shall be decrypted.
	cipherTextLength	Contains the length of the cipher text in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	plainTextLengthPtr	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 plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
Parameters (out):	plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.

Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to feed the symmetrical decryption service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The decryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00667]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymDecryptUpdate.

8.3.6.6 Cal_SymDecryptFinish

[SWS_Cal_00243] ⌈

Service name:	Cal_SymDecryptFinish	

Syntax:	`Cal_ReturnType Cal_SymDecryptFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymDecryptCtxBufType contextBuffer,`
	`    uint8* plainTextPtr,`
	`    uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x1B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical decryption computation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	plainTextLengthPtr	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 plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
Parameters (out):	plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to finish the symmetrical decryption service. If the service state given by the context buffer is "idle", the function has to return	

	with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The decryption process is done by the underlying primitive.

⌋ ()

[SWS_Cal_00668]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymDecryptFinish.

8.3.7 Asymmetrical interface

Asymmetric-key algorithms are algorithms that use pairs of cryptographic keys (public and private keys) for decryption and encryption. The private key, in practice, represent a secret while the public key can be made publically available.

8.3.7.1 Cal_AsymEncryptStart

[SWS_Cal_00250] ⌈

Service name:	Cal_AsymEncryptStart	

Syntax:	`Cal_ReturnType Cal_AsymEncryptStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymEncryptCtxBufType contextBuffer,`
	`    const Cal_AsymPublicKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x1C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the asymmetrical encryption computation.
	keyPtr	Holds a pointer to the key which has to be used during the asymmetrical encryption computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the asymmetrical encrypt service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymEncryptStart.

8.3.7.2 Cal_AsymEncryptUpdate

[SWS_Cal_00256] ⌈

Service name:	Cal_AsymEncryptUpdate	

Syntax:	`Cal_ReturnType Cal_AsymEncryptUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymEncryptCtxBufType contextBuffer,`
	`    const uint8* plainTextPtr,`
	`    uint32 plainTextLength,`
	`    uint8* cipherTextPtr,`
	`    uint32* cipherTextLengthPtr`
	`)`	

Service ID[hex]:	0x1D	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the asymmetrical encryption computation.
	plainTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
	plainTextLength	Contains the length of the plain text in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	cipherTextLengthPtr	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 cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
Parameters (out):	cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to feed the asymmetrical encryption service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The encryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00669]

⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CAL_E_SMALL_BUFFER shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymEncryptUpdate.

8.3.7.3 Cal_AsymEncryptFinish

[SWS_Cal_00265] ⌈

Service name:	Cal_AsymEncryptFinish	

Syntax:	`Cal_ReturnType Cal_AsymEncryptFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymEncryptCtxBufType contextBuffer,`
	`    uint8* cipherTextPtr,`
	`    uint32* cipherTextLengthPtr`
	`)`	

Service ID[hex]:	0x1E	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of th CAL module configuration which has to be used during the asymmetrical encryption computation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	cipherTextLengthPtr	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 cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
Parameters (out):	cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to finish the asymmetrical encryption service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The encryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00670]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CAL_E_SMALL_BUFFER shall be returned. ⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymEncryptFinish.

8.3.7.4 Cal_AsymDecryptStart

[SWS_Cal_00272] ⌈

Service name:	Cal_AsymDecryptStart	

Syntax:	`Cal_ReturnType Cal_AsymDecryptStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymDecryptCtxBufType contextBuffer,`
	`    const Cal_AsymPrivateKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x1F	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the asymmetrical decryption computation.
	keyPtr	Holds a pointer to the key which has to be used during the asymmetrical encryption computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the asymmetrical decrypt service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymDecryptStart.

8.3.7.5 Cal_AsymDecryptUpdate

[SWS_Cal_00278] ⌈

Service name:	Cal_AsymDecryptUpdate	

Syntax:	`Cal_ReturnType Cal_AsymDecryptUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymDecryptCtxBufType contextBuffer,`
	`    const uint8* cipherTextPtr,`
	`    uint32 cipherTextLength,`
	`    uint8* plainTextPtr,`
	`    uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x20	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the asymmetrical decryption computation.
	cipherTextPtr	Holds a pointer to the encrypted data.
	cipherTextLength	Contains the length of the encrypted data in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

	plainTextLengthPtr	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 plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
Parameters (out):	plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to feed the asymmetrical decryption service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The decryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00671]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymDecryptUpdate.

8.3.7.6 Cal_AsymDecryptFinish

[SWS_Cal_00287] ⌈

Service name:	Cal_AsymDecryptFinish	

Syntax:	`Cal_ReturnType Cal_AsymDecryptFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymDecryptCtxBufType contextBuffer,`
	`    uint8* plainTextPtr,`
	`    uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x21	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the asymmetrical computation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	plainTextLengthPtr	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 plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
Parameters (out):	plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.

Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to finish the asymmetrical decryption service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The decryption process is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00672]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymDecryptFinish.

8.3.8 Signature interface

A digital signature is a type of asymmetric cryptography. Digital signatures are equivalent to traditional handwritten signatures in many respects.
Digital signatures can be used to authenticate the source of messages as well as to prove integrity of signed messages. If a message is digitally signed, any change in the message after signature will invalidate the signature. Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature.

8.3.8.1 Cal_SignatureGenerateStart

[SWS_Cal_00294] ⌈

Service name:	Cal_SignatureGenerateStart	

Syntax:	`Cal_ReturnType Cal_SignatureGenerateStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SignatureGenerateCtxBufType contextBuffer,`
	`    const Cal_AsymPrivateKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature generation.
	keyPtr	Holds a pointer to the key necessary for the signature generation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the signature generate service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SignatureGenerateStart.

8.3.8.2 Cal_SignatureGenerateUpdate

[SWS_Cal_00300] ⌈

Service name:	Cal_SignatureGenerateUpdate	

Syntax:	`Cal_ReturnType Cal_SignatureGenerateUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SignatureGenerateCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x23	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature generation.
	dataPtr	Holds a pointer to the data that shall be signed.
	dataLength	Contains the length of the data to be signed.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the signature generation service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The signature computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SignatureGenerateUpdate.

8.3.8.3 Cal_SignatureGenerateFinish

[SWS_Cal_00307] ⌈

Service name:	Cal_SignatureGenerateFinish	

Syntax:	`Cal_ReturnType Cal_SignatureGenerateFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SignatureGenerateCtxBufType contextBuffer,`
	`    uint8* resultPtr,`
	`    uint32* resultLengthPtr`
	`)`	

Service ID[hex]:	0x24	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature generation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	resultLengthPtr	Holds a pointer to the 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. On returning from this function the actual length of the computed signature shall be stored
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the signature generation.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result
Description:	This function shall be used to finish the signature generation service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The signature computation is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00673]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SignatureGenerateFinish.

8.3.8.4 Cal_SignatureVerifyStart

[SWS_Cal_00314] ⌈

Service name:	Cal_SignatureVerifyStart

Syntax:	`Cal_ReturnType Cal_SignatureVerifyStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SignatureVerifyCtxBufType contextBuffer,`
	`    const Cal_AsymPublicKeyType* keyPtr`
	`)`

Service ID[hex]:	0x25
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature computation/verification.
	keyPtr	Holds a pointer to the key necessary for the signature verification.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the signature verify service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SignatureVerifyStart.

8.3.8.5 Cal_SignatureVerifyUpdate

[SWS_Cal_00320] ⌈

Service name:	Cal_SignatureVerifyUpdate	

Syntax:	`Cal_ReturnType Cal_SignatureVerifyUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SignatureVerifyCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x26	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature computation/verification.
	dataPtr	Holds a pointer to the signature which shall be verified.
	dataLength	Contains the length of the signature to verify in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the signature verification service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The signature computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SignatureVerifyUpdate.

8.3.8.6 Cal_SignatureVerifyFinish

[SWS_Cal_00327] ⌈

Service name:	Cal_SignatureVerifyFinish	
Syntax:	`Cal_ReturnType Cal_SignatureVerifyFinish( Cal_ConfigIdType cfgId, Cal_SignatureVerifyCtxBufType contextBuffer, const uint8* signaturePtr, uint32 signatureLength, Cal_VerifyResultType* resultPtr )`	
Service ID[hex]:	0x27	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature computation/verification.
	signaturePtr	Holds a pointer to the memory location which holds the signature to be verified.
	signatureLength	Holds the length of the Signature to be verified.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the signature verification.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the signature verification service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The signature computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SignatureVerifyFinish.

8.3.9 Compression / Decompression interface

Due to usage of compression/decompression algorithms it is possible to reduce of the amount of data, which must be processed by encryption/decryption. Due to appropriate seletion of the compression/decompression algorithm, the aggregated load can be reduced: the compression and encryption of the reduced amount of data respectively decription and decompression consumes fewer resources than the encryption and decryption of the uncompressed data.

The following APIs can be used for compression and decompression of data.

8.3.9.1 Cal_CompressStart

[SWS_Cal_00756]⌈

Service name:	Cal_CompressStart	

Syntax:	`Cal_ReturnType Cal_CompressStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_CompressCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x4d	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the compression computation
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This function shall be used to initialize the compression service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_CompressStart.

8.3.9.2 Cal_CompressUpdate

[SWS_Cal_00757]⌈

Service name:	Cal_CompressUpdate	

Syntax:	`Cal_ReturnType Cal_CompressUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_CompressCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x4e	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the compression computation
	dataPtr	Holds a pointer to the data that shall be compressed.
	dataLength	Contains the number of the data in bytes to be compressed
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result
Description:	This function shall be used to feed the compression service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The compression computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_CompressUpdate.

8.3.9.3 Cal_CompressFinish

[SWS_Cal_00758]⌈

Service name:	Cal_CompressFinish	

Syntax:	`Cal_ReturnType Cal_CompressFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_CompressCtxBufType contextBuffer,`
	`    uint8* resultPtr,`
	`    uint32* resultLengthPtr`
	`)`	

Service ID[hex]:	0x4f	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the compression computation
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	resultLengthPtr	Holds a pointer to the 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. On returning from this function, the actual length of the compression shall be stored
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the compression.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result
Description:	This function shall be used to finish the compression service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the	

	state of this service to "idle", and store this state in the context buffer. The compression computation is done by the underlying primitive.

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_CompressFinish.

8.3.9.4 Cal_DecompressStart

[SWS_Cal_00759]⌈

Service name:	Cal_DecompressStart	

Syntax:	`Cal_ReturnType Cal_DecompressStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_DecompressCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x50	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the decompression computation
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This function shall be used to initialize the decompression service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_DecompressStart.

8.3.9.5 Cal_DecompressUpdate

[SWS_Cal_00760]⌈

Service name:	Cal_DecompressUpdate

Syntax:	`Cal_ReturnType Cal_DecompressUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_DecompressCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`

Service ID[hex]:	0x51
Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the decompression computation
	dataPtr	Holds a pointer to the data that shall be decompressed.
	dataLength	Contains the number of the data in bytes to be decompressed
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result
Description:	This function shall be used to feed the decompression service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The decompression computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_DecompressUpdate.

8.3.9.6 Cal_DecompressFinish

[SWS_Cal_00761]⌈

Service name:	Cal_DecompressFinish	

Syntax:	`Cal_ReturnType Cal_DecompressFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_DecompressCtxBufType contextBuffer,`
	`    uint8* resultPtr,`
	`    uint32* resultLengthPtr`
	`)`	

Service ID[hex]:	0x52	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the decompression computation
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	resultLengthPtr	Holds a pointer to the 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. On returning from this function, the actual length of the decompression shall be stored
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the decompression.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result
Description:	This function shall be used to finish the decompression service.	

	If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The decompression computation is done by the underlying primitive.

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_DecompressFinish.

8.3.10 Checksum interface

The goal of checksum algorithms is to detect accidental modification such as corruption to stored data or errors in a communication channel. They are not designed to detect intentional corruption by a malicious agent. Indeed, many checksum algorithms can be easily inverted, in the sense that one can easily modify the data so as to preserve its checksum.

8.3.10.1 Cal_ChecksumStart

[SWS_Cal_00335]⌈

Service name:	Cal_ChecksumStart	

Syntax:	`Cal_ReturnType Cal_ChecksumStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_ChecksumCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x28	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the checksum computation.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the checksum service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_ChecksumStart.

8.3.10.2 Cal_ChecksumUpdate

[SWS_Cal_00341] ⌈

Service name:	Cal_ChecksumUpdate	

Syntax:	`Cal_ReturnType Cal_ChecksumUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_ChecksumCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x29	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the checksum computation.
	dataPtr	Holds a pointer to the data for which the checksum shall be calculated.
	dataLength	Contains the length of the input data in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the checksum service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The checksum computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_ChecksumUpdate.

8.3.10.3 Cal_ChecksumFinish

[SWS_Cal_00348] ⌈

Service name:	Cal_ChecksumFinish	

Syntax:	`Cal_ReturnType Cal_ChecksumFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_ChecksumCtxBufType contextBuffer,`
	`    uint8* resultPtr,`
	`    uint32* resultLengthPtr,`
	`    boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x2A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the checksum computation.
	TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: Truncation is allowed.

		FALSE: Truncation is not allowed.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	resultLengthPtr	Holds a pointer to the 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. On returning from this function the actual length of the computed checksum shall be stored
Parameters (out):	resultPtr	Holds a pointer to the memory location which will hold the result of the checksum calculation. If the result does not fit into the given buffer, the result shall be truncated.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result, and truncation was not allowed.
Description:	This function shall be used to finish the checksum service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The checksum computation is done by the underlying primitive.	

⌋ ()

[SWS_Cal_00674]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small and truncation is allowed, the result of the computation shall be truncated to the size of the provided buffer, and `CAL_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_ChecksumFinish.

8.3.11 Key derivation interface

In cryptography, a key derivation function (or KDF) is a function which derives one or more secret keys from a secret value and/or other known information such as a passphrase or cryptographic key.

8.3.11.1 Cal_KeyDeriveStart

[SWS_Cal_00355] ⌈

Service name:	Cal_KeyDeriveStart

Syntax:	`Cal_ReturnType Cal_KeyDeriveStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_KeyDeriveCtxBufType contextBuffer,`
	`    uint32 keyLength,`
	`    uint32 iterations`
	`)`

Service ID[hex]:	0x2B

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key derivation.
	keyLength	Holds the length of the key to be derived by the underlying key derivation primitive.
	iterations	Holds the number of iterations to be performed by the underlying key derivation primitive.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the key derivation service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_KeyDeriveStart.

8.3.11.2 Cal_KeyDeriveUpdate

[SWS_Cal_00362] ⌈

Service name:	Cal_KeyDeriveUpdate	

Syntax:	`Cal_ReturnType Cal_KeyDeriveUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_KeyDeriveCtxBufType contextBuffer,`
	`    const uint8* passwordPtr,`
	`    uint32 passwordLength,`
	`    const uint8* saltPtr,`
	`    uint32 saltLength`
	`)`	

Service ID[hex]:	0x2C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key derivation.
	passwordPtr	Holds a pointer to the password, i.e. the original key, from which to derive a new key.
	passwordLength	Holds the length of the password in bytes.
	saltPtr	Holds a pointer to the cryptographic salt, i.e. a random number, for the underlying primitive.
	saltLength	Holds the length of the salt in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed

Description:	This function shall be used to feed the key derivation service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The key derivation computation is done by the underlying primitive.

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_KeyDeriveUpdate.

8.3.11.3 Cal_KeyDeriveFinish

[SWS_Cal_00371] ⌈

Service name:	Cal_KeyDeriveFinish	

Syntax:	`Cal_ReturnType Cal_KeyDeriveFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_KeyDeriveCtxBufType contextBuffer,`
	`    Cal_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x2D	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key derivation.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	keyPtr	Holds a pointer to the memory location which will hold the result of the key derivation.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the key generation service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The key derivation computation is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_KeyDeriveFinish.

8.3.12 Key exchange interface

Two users that each have a private secret can use a key exchange protocol to obtain a common secret, e.g. a key for a symmetric-key algorithm, without telling each other their private secret and without any listener being able to obtain the common secret or their private secrets.

8.3.12.1 Cal_KeyExchangeCalcPubVal

[SWS_Cal_00377] ⌈

Service name:	Cal_KeyExchangeCalcPubVal	

Syntax:	`Cal_ReturnType Cal_KeyExchangeCalcPubVal(`
	`    Cal_ConfigIdType cfgId,`
	`    const Cal_KeyExchangeBaseType* basePtr,`
	`    const Cal_KeyExchangePrivateType* privateValuePtr,`
	`    uint8* publicValuePtr,`
	`    uint32* publicValueLengthPtr`
	`)`	

Service ID[hex]:	0x2E	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration that has to be used during the key exchange.
	basePtr	Holds a pointer to the base information known to both users of the key exchange protocol.
	privateValuePtr	Holds a pointer to the private information known only to the current user of the key exchange protocol.
Parameters (inout):	publicValueLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by publicValuePtr. On returning from this function the actual length of the calculated public value shall be stored.
Parameters (out):	publicValuePtr	Holds a pointer to the memory location which will hold the public value of the key exchange protocol.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result.
Description:	This function shall be used to start the public value calculation service of the CAL module. The function shall call the function Cpl_<Primitive> of the primitive which is identified by the "cfgId" and return the value returned by that function.	

⌋ ()

[SWS_Cal_00675]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_KeyExchangeCalcPubVal.

8.3.12.2 Cal_KeyExchangeCalcSecretStart

[SWS_Cal_00396] ⌈

Service name:	Cal_KeyExchangeCalcSecretStart

Syntax:	`Cal_ReturnType Cal_KeyExchangeCalcSecretStart(`
	`    Cal_ConfigIdType cfgId,`

	`    Cal_KeyExchangeCalcSecretCtxBufType contextBuffer,`
	`    const Cal_KeyExchangeBaseType* basePtr,`
	`    const Cal_KeyExchangePrivateType* privateValuePtr`
	`)`	

Service ID[hex]:	0x2F	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration that has to be used during the key exchange.
	basePtr	Holds a pointer to the base information known to both users of the key exchange protocol.
	privateValuePtr	Holds a pointer to the private information known only to the current user of the key exchange protocol.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the key exchange service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_KeyExchangeCalcSecretStart.

8.3.12.3 Cal_KeyExchangeCalcSecretUpdate

[SWS_Cal_00404] ⌈

Service name:	Cal_KeyExchangeCalcSecretUpdate	

Syntax:	`Cal_ReturnType Cal_KeyExchangeCalcSecretUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_KeyExchangeCalcSecretCtxBufType contextBuffer,`
	`    const uint8* partnerPublicValuePtr,`
	`    uint32 partnerPublicValueLength`
	`)`	

Service ID[hex]:	0x30	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration that has to be used during the key exchange.
	partnerPublicValuePtr	Holds a pointer to the data representing the public value of the key exchange partner.
	partnerPublicValueLength	Contains the length of the part of the partner value in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed

Description:	This function shall be used to feed the key exchange service with the public value coming from the partner of the key exchange protocol. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the shared secret is done by the underlying primitive.

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_KeyExchangeCalcSecretUpdate.

8.3.12.4 Cal_KeyExchangeCalcSecretFinish

[SWS_Cal_00411]⌈

Service name:	Cal_KeyExchangeCalcSecretFinish	

Syntax:	`Cal_ReturnType Cal_KeyExchangeCalcSecretFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_KeyExchangeCalcSecretCtxBufType contextBuffer,`
	`    uint8* sharedSecretPtr,`
	`    uint32* sharedSecretLengthPtr,`
	`    boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x31	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration that has to be used during the key exchange.
	TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: Truncation is allowed. FALSE: Truncation is not allowed.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	sharedSecretLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by sharedSecretPtr. On returning from this function the actual length of the computed value shall be stored.
Parameters (out):	sharedSecretPtr	Holds a pointer to the memory location which will hold the result of the key exchange. If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed CAL_E_SMALL_BUFFER: The provided buffer is too small to store the result, and truncation was not allowed.
Description:	This function shall be used to finish the key exchange service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the	

	primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the shared secret is done by the underlying primitive.

⌋ ()

[SWS_Cal_00676]
⌈ The CAL shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small and truncation is allowed, the result of the computation shall be truncated to the size of the provided buffer, and `CAL_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CAL_E_SMALL_BUFFER` shall be returned. ⌋ ()
Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_KeyExchangeCalcSecretFinish.

8.3.13 Symmetrical key extract interface

Symmetrical key extract interface is used to extract a symmetrical key structure from certain data sources.
Note that this interface may be used for key transport purposes. In this case, any necessary auxiliary information (e.g., wrapping key, shared information, randomness) will have to be encoded unambiguously into the data provided in the dataPtr buffer.

8.3.13.1 Cal_SymKeyExtractStart

[SWS_Cal_00418] ⌈

Service name:	Cal_SymKeyExtractStart	

Syntax:	`Cal_ReturnType Cal_SymKeyExtractStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyExtractCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x32	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the symmetrical key extraction service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymKeyExtractStart.

8.3.13.2 Cal_SymKeyExtractUpdate

[SWS_Cal_00425] ⌈

Service name:	Cal_SymKeyExtractUpdate	

Syntax:	`Cal_ReturnType Cal_SymKeyExtractUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyExtractCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x33	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
	dataPtr	Holds a pointer to the data which contains the key in a format which cannot be used directly by the CAL. From this data the key will be extracted in a CAL-conforming format.
	dataLength	Holds the length of the data in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the symmetrical key extraction service with input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the extraction algorithm is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_SymKeyExtractUpdate.

8.3.13.3 Cal_SymKeyExtractFinish

[SWS_Cal_00432] ⌈

Service name:	Cal_SymKeyExtractFinish	

Syntax:	`Cal_ReturnType Cal_SymKeyExtractFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyExtractCtxBufType contextBuffer,`
	`    Cal_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x34	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which

		has to be used during the key extraction.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	keyPtr	Holds a pointer to a structure where the result (i.e. the symmetrical key) is stored in.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the symmetrical key extraction service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the extraction algorithm is done by the underlying primitive.	

⌋ ()

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the funcion Cal_SymKeyExtractFinish.

8.3.14 Symmetrical key wrapping interface

Symmetrical key wrapping interface is used to export a symmetrical key structure, e.g. to be used on a different device. To be able to use symmetric and asymmetric wrapping keys, two different interfaces are standardised.

8.3.14.1 Cal_SymKeyWrapSymStart

[SWS_Cal_00744] ⌈

Service name:	Cal_SymKeyWrapSymStart	

Syntax:	`Cal_ReturnType Cal_SymKeyWrapSymStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyWrapSymCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr,`
	`    const Cal_SymKeyType* wrappingKeyPtr`
	`)`	

Service ID[hex]:	0x3c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
	keyPtr	Holds a pointer to the symmetric key to be wrapped.
	wrappingKeyPtr	Holds a pointer to the key used for wrapping.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to initialize the symmetrical key wrapping service of the CAL module.	

	The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.14.2 Cal_SymKeyWrapSymUpdate

[SWS_Cal_00745] ⌈

Service name:	Cal_SymKeyWrapSymUpdate	

Syntax:	`Cal_ReturnType Cal_SymKeyWrapSymUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyWrapSymCtxBufType contextBuffer,`
	`    uint8* dataPtr,`
	`    uint32* dataLengthPtr`
	`)`	

Service ID[hex]:	0x3d	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	dataLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by dataPtr. When the request has finished, the actual length of the computed value shall be stored.
Parameters (out):	dataPtr	Holds a pointer to the memory location which will hold the first chunk of the result of the key wrapping. If the result does not fit into the given buffer, the caller shall call the service again, until *dataLengthPtr is equal to zero, indicating that the complete result has been retrieved.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to retrieve the result of the key wrapping operation from the symmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.14.3 Cal_SymKeyWrapSymFinish

[SWS_Cal_00746] ⌈

Service name:	Cal_SymKeyWrapSymFinish	

Syntax:	`Cal_ReturnType Cal_SymKeyWrapSymFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyWrapSymCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x3e	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to finish the symmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.14.4 Cal_SymKeyWrapAsymStart

[SWS_Cal_00747] ⌈

Service name:	Cal_SymKeyWrapAsymStart	

Syntax:	`Cal_ReturnType Cal_SymKeyWrapAsymStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyWrapAsymCtxBufType contextBuffer,`
	`    const Cal_SymKeyType* keyPtr,`
	`    const Cal_AsymPublicKeyType* wrappingKeyPtr`
	`)`	

Service ID[hex]:	0x3f	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration which has to be used during the key wrapping.
	keyPtr	Holds a pointer to the symmetric key to be wrapped.
	wrappingKeyPtr	Holds a pointer to the public key used for wrapping.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to initialize the symmetrical key wrapping service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and	

	return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.14.5 Cal_SymKeyWrapAsymUpdate

[SWS_Cal_00748] ⌈

Service name:	Cal_SymKeyWrapAsymUpdate	

Syntax:	`Cal_ReturnType Cal_SymKeyWrapAsymUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyWrapAsymCtxBufType contextBuffer,`
	`    uint8* dataPtr,`
	`    uint32* dataLengthPtr`
	`)`	

Service ID[hex]:	0x40	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	dataLengthPtr	Holds a pointer to the memory location in which the length information is stored.
Parameters (out):	dataPtr	Holds a pointer to the memory location which will hold the first chunk of the result of the key wrapping. If the result does not fit into the given buffer, the caller shall call the service again, until *dataLengthPtr is equal to zero, indicating that the complete result has been retrieved.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to retrieve the result of the key wrapping operation from the symmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.14.6 Cal_SymKeyWrapAsymFinish

[SWS_Cal_00749] ⌈

Service name:	Cal_SymKeyWrapAsymFinish

Syntax:	`Cal_ReturnType Cal_SymKeyWrapAsymFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_SymKeyWrapAsymCtxBufType contextBuffer`
	`)`

Service ID[hex]:	0x41

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to finish the symmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.15 Asymmetrical key extract interfaces

Asymmetrical key extract interface is used to extract an asymmetrical key structure (e.g. public and private key pair) from certain data sources.
Note that this interface may be used for key transport purposes. In this case, any necessary auxiliary information (e.g., wrapping key, shared information, randomness) will have to be encoded unambiguously into the data provided in the dataPtr buffer.

8.3.15.1 Cal_AsymPublicKeyExtractStart

[SWS_Cal_00436]⌈

Service name:	Cal_AsymPublicKeyExtractStart	

Syntax:	`Cal_ReturnType Cal_AsymPublicKeyExtractStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPublicKeyExtractCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x35	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the asymmetrical public key extraction service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the	

	function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymPublicKeyExtractStart. ⌋ ()

8.3.15.2 Cal_AsymPublicKeyExtractUpdate

[SWS_Cal_00443]⌈

Service name:	Cal_AsymPublicKeyExtractUpdate	

Syntax:	`Cal_ReturnType Cal_AsymPublicKeyExtractUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPublicKeyExtractCtxBufType contextBuffer,`
	`    const uint8* dataPtr,`
	`    uint32 dataLength`
	`)`	

Service ID[hex]:	0x36	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
	dataPtr	Holds a pointer to the data which contains the key in a format which cannot be used directly by the CAL. From this data the key will be extracted in a CAL-conforming format.
	dataLength	Holds the length of the data in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the asymmetrical public key extraction service with input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the extraction algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymPublicKeyExtractUpdate. ⌋ ()

8.3.15.3 Cal_AsymPublicKeyExtractFinish

[SWS_Cal_00450]⌈

Service name:	Cal_AsymPublicKeyExtractFinish

Syntax:	`Cal_ReturnType Cal_AsymPublicKeyExtractFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPublicKeyExtractCtxBufType contextBuffer,`
	`    Cal_AsymPublicKeyType* keyPtr`

	)	
Service ID[hex]:	0x37	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	keyPtr	Holds a pointer to a structure where the result (i.e. the symmetrical key) is stored in.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the asymmetrical public key extraction service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the extraction algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymPublicKeyExtractFinish. ⌋
()

8.3.15.4 Cal_AsymPrivateKeyExtractStart

[SWS_Cal_00680]⌈

Service name:	Cal_AsymPrivateKeyExtractStart	

Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyExtractStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPrivateKeyExtractCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x38	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to initialize the asymmetrical private key extraction service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymPrivateKeyExtractStart. ⌋ ()

8.3.15.5 Cal_AsymPrivateKeyExtractUpdate

[SWS_Cal_00682] ⌈

Service name:	Cal_AsymPrivateKeyExtractUpdate	
Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyExtractUpdate( Cal_ConfigIdType cfgId, Cal_AsymPrivateKeyExtractCtxBufType contextBuffer, const uint8* dataPtr, uint32 dataLength )`	
Service ID[hex]:	0x39	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
	dataPtr	Holds a pointer to the data which contains the key in a format which cannot be used directly by the CAL. From this data the key will be extracted in a CAL-conforming format.
	dataLength	Holds the length of the data in bytes.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to feed the asymmetrical private key extraction service with input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the extraction algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymPrivateKeyExtractUpdate. ⌋ ()

8.3.15.6 Cal_AsymPrivateKeyExtractFinish

[SWS_Cal_00684] ⌈

Service name:	Cal_AsymPrivateKeyExtractFinish
Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyExtractFinish( Cal_ConfigIdType cfgId, Cal_AsymPrivateKeyExtractCtxBufType contextBuffer, Cal_AsymPrivateKeyType* keyPtr )`
Service ID[hex]:	0x3A
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	keyPtr	Holds a pointer to a structure where the result (i.e. the symmetrical key) is stored in.
Return value:	Cal_ReturnType	CAL_E_OK: Request successful CAL_E_NOT_OK: Request failed
Description:	This function shall be used to finish the asymmetrical private key extraction service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the extraction algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable to the function Cal_AsymPrivateKeyExtractFinish. ⌋ ()

8.3.16 Asymmetrical key wrapping interface

Asymmetrical key wrapping interface is used to export a (asymmetric) private key structure, e.g. to be used on a different device. To be able to use symmetric and asymmetric wrapping keys, two different interfaces are standardised.

8.3.16.1 Cal_AsymPrivateKeyWrapSymStart

[SWS_Cal_00750] ⌈

Service name:	Cal_AsymPrivateKeyWrapSymStart	

Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyWrapSymStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPrivateKeyWrapSymCtxBufType contextBuffer,`
	`    const Cal_AsymPrivateKeyType* keyPtr,`
	`    const Cal_SymKeyType* wrappingKeyPtr`
	`)`	

Service ID[hex]:	0x42	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
	keyPtr	Holds a pointer to the private key to be wrapped.
	wrappingKeyPtr	Holds a pointer to the key used for wrapping.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: request successful

		CAL_E_NOT_OK: request failed
Description:	This interface shall be used to initialize the asymmetrical key wrapping service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.16.2 Cal_AsymPrivateKeyWrapSymUpdate

[SWS_Cal_00751] ⌈

Service name:	Cal_AsymPrivateKeyWrapSymUpdate	

Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyWrapSymUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPrivateKeyWrapSymCtxBufType contextBuffer,`
	`    uint8* dataPtr,`
	`    uint32* dataLengthPtr`
	`)`	

Service ID[hex]:	0x43	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	dataLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by dataPtr.
Parameters (out):	dataPtr	Holds a pointer to the memory location which will hold the first chunk of the result of the key wrapping. If the result does not fit into the given buffer, the caller shall call the service again, until *dataLengthPtr is equal to zero, indicating that the complete result has been retrieved.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to retrieve the result of the key wrapping operation from the asymmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.16.3 Cal_AsymPrivateKeyWrapSymFinish

[SWS_Cal_00752] ⌈

Service name:	Cal_AsymPrivateKeyWrapSymFinish	

Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyWrapSymFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPrivateKeyWrapSymCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x44	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to finish the asymmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.16.4 Cal_AsymPrivateKeyWrapAsymStart

[SWS_Cal_00753] ⌈

Service name:	Cal_AsymPrivateKeyWrapAsymStart	

Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyWrapAsymStart(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPrivateKeyWrapAsymCtxBufType contextBuffer,`
	`    const Cal_AsymPrivateKeyType* keyPtr,`
	`    const Cal_AsymPublicKeyType* wrappingKeyPtr`
	`)`	

Service ID[hex]:	0x45	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration which has to be used during the key wrapping.
	keyPtr	Holds a pointer to the private key to be wrapped.
	wrappingKeyPtr	Holds a pointer to the public key used for wrapping.
Parameters (inout):	None	
Parameters (out):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return value:	Cal_ReturnType	CAL_E_OK: request successful

		CAL_E_NOT_OK: request failed
Description:	This interface shall be used to initialize the asymmetrical key wrapping service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.16.5 Cal_AsymPrivateKeyWrapAsymUpdate

[SWS_Cal_00754]⌈

Service name:	Cal_AsymPrivateKeyWrapAsymUpdate	

Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyWrapAsymUpdate(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPrivateKeyWrapAsymCtxBufType contextBuffer,`
	`    uint8* dataPtr,`
	`    uint32* dataLengthPtr`
	`)`	

Service ID[hex]:	0x46	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
	dataLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by dataPtr. When the request has finished, the actual length of the computed value shall be stored.
Parameters (out):	dataPtr	Holds a pointer to the memory location which will hold the first chunk of the result of the key wrapping. If the result does not fit into the given buffer, the caller shall call the service again, until *dataLengthPtr is equal to zero, indicating that the complete result has been retrieved.
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to retrieve the result of the key wrapping operation from the asymmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.3.16.6 Cal_AsymPrivateKeyWrapAsymFinish

[SWS_Cal_00755]⌈

Service name:	Cal_AsymPrivateKeyWrapAsymFinish	

Syntax:	`Cal_ReturnType Cal_AsymPrivateKeyWrapAsymFinish(`
	`    Cal_ConfigIdType cfgId,`
	`    Cal_AsymPrivateKeyWrapAsymCtxBufType contextBuffer`
	`)`	

Service ID[hex]:	0x47	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CAL module configuration which has to be used during the key wrapping.
Parameters (inout):	contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Parameters (out):	None	
Return value:	Cal_ReturnType	CAL_E_OK: request successful CAL_E_NOT_OK: request failed
Description:	This interface shall be used to finish the asymmetrical key wrapping service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The calculation of the wrapping algorithm is done by the underlying primitive.	

Regarding error detection, the requirements SWS_Cal_00064, SWS_Cal_00488 and SWS_Cal_00489 are applicable. ⌋ ()

8.4 Dependencies to cryptographic library API functions

8.4.1 Types for the Cryptographic Primitives

8.4.1.1 Cpl_<Primitive>ConfigType

[SWS_Cal_00544]⌈

Name:	`Cpl_<Primitive>_ConfigType`
Type:	`Structure`
Range:	Implementation specific.
Description:	Data structure which shall encompass all information needed to specify the information needed for the <Primitive> cryptographic primitive.

⌋ ()

8.4.2 API functions of the cryptographic primitives

[SWS_Cal_00461]

⌈ For every API function of a cryptographic service, the corresponding cryptographic primitive shall contain a corresponding function. ⌋ (SRS_Csm_00006)

[SWS_Cal_00505]

⌈ The implementation of the basic cryptographic routines shall be synchronous and reentrant. ⌋ ()

8.4.2.1 Cpl_<Primitive>Start

[SWS_Cal_00701] ⌈

Service name:	Cpl_<Primitive>Start	

Syntax:	`Cal_ReturnType Cpl_<Primitive>Start(`
	`    <type> <xxx>,`
	`    <type> <yyy>,`
	`    <type> <zzz>`
	`)`	

Service ID[hex]:	--	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Cal_<Service>Start(), with the exception of the argument cfgId. This argument is of type â€œCal_ConfigIdTypeâ€œ in Cal_<Service>Start(). In Cpl_<Primitive>Start the argument cfgId shall be replaced by an argument cfgPtr of type â€œconst void *â€œ.
Parameters (inout):	<yyy>	The arguments <yyy> shall be identical to the arguments of the corresponding function Cal_<Service>Start().
Parameters (out):	<zzz>	The arguments <zzz> shall be identical to the arguments of the corresponding function Cal_<Service>Start().
Return value:	Cal_ReturnType	The return values shall be identical to those of the corresponding function Cal_<Service>Start().
Description:	This function shall initialize the computation of the cryptographic primitive, so that the primitive is able to process input data. Intermediate results, that are required for further processing of the service, shall be stored in the context buffer, which is given as an argument of this function.	

⌋ ()

The API "Cpl_<Primitive>Start" has a parameter "cfgPtr" of type "const void *".
When calling this API, the parameter "cfgPtr" shall point to a constant variable of type "Cpl_<Primitive>ConfigType", but shall be cast to "const void *".
Reason for this is to have a common definition of the parameter list of this API for all primitives of one service, because in the structure Cal_<Service>ConfigType one element is a function pointer to this API.

8.4.2.2 Cpl_<Primitive>Update

[SWS_Cal_00702] ⌈

Service name:	Cpl_<Primitive>Update

Syntax:	`Cal_ReturnType Cpl_<Primitive>Update(`
	`    <type> <xxx>,`
	`    <type> <yyy>,`
	`    <type> <zzz>`
	`)`

Service ID[hex]:	--	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Cal_<Service>Update(), with the exception of the argument cfgId. This argument is of type â€œCal_ConfigIdTypeâ€œ in Cal_<Service>Update(). In Cpl_<Primitive>Update the argument cfgId shall be replaced by an argument cfgPtr of type â€œconst void *â€œ.
Parameters (inout):	<yyy>	The arguments <yyy> shall be identical to the arguments of the corresponding function Cal_<Service>Update().
Parameters (out):	<zzz>	The arguments <zzz> shall be identical to the arguments of the corresponding function Cal_<Service>Update().
Return value:	Cal_ReturnType	The return values shall be identical to those of the corresponding function Cal_<Service>Update().
Description:	This function shall process a chunk of the given input data with the algorithm of the cryptographic primitive. Intermediate results, that are derived from previous processing steps of this service, have to be taken from the context buffer, which is given as an argument of this function. Intermediate results, that are required for further processing of the service, shall be stored in the context buffer, which is given as an argument of this function.	

⌋ ()
The API "Cpl_<Primitive>Update" has a parameter "cfgPtr" of type "const void *". When calling this API, the parameter "cfgPtr" shall point to a constant variable of type "Cpl_<Primitive>ConfigType", but shall be cast to "const void *".
Reason for this is to have a common definition of the parameter list of this API for all primitives of one service, because in the structure Cal_<Service>ConfigType one element is a function pointer to this API.

8.4.2.3 Cpl_<Primitive>Finish

[SWS_Cal_00703] ⌈

Service name:	Cpl_<Primitive>Finish	

Syntax:	`Cal_ReturnType Cpl_<Primitive>Finish(`
	`    <type> <xxx>,`
	`    <type> <yyy>,`
	`    <type> <zzz>`
	`)`	

Service ID[hex]:	--	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Cal_<Service>Finish(), with the exception of the argument cfgId. This argument is of type â€œCal_ConfigIdTypeâ€œ in Cal_<Service>Finish(). In Cpl_<Primitive>Finish the argument cfgId shall be replaced by an argument cfgPtr of type â€œconst void *â€œ.
Parameters (inout):	<yyy>	The arguments <yyy> shall be identical to the arguments of the corresponding function Cal_<Service>Finish().
Parameters (out):	<zzz>	The arguments <zzz> shall be identical to the arguments of the corresponding function Cal_<Service>Finish().
Return value:	Cal_ReturnType	The return values shall be identical to those of the corresponding function Cal_<Service>Finish().

Description:	This function shall finish the computation of the cryptographic primitive and store the result into the memory location given. Intermediate results, that are derived from previous processing steps of this service, have to be taken from the context buffer, which is given as an argument of this function.

⌋ ()

The API "Cpl_<Primitive>Finish" has a parameter "cfgPtr" of type "const void *".
When calling this API, the parameter "cfgPtr" shall point to a constant variable of type "Cpl_<Primitive>ConfigType", but shall be cast to "const void *".
Reason for this is to have a common definition of the parameter list of this API for all primitives of one service, because in the structure Cal_<Service>ConfigType one element is a function pointer to this API.

8.4.2.4 Cpl_<Primitive>

[SWS_Cal_00704] ⌈

Service name:	Cpl_<Primitive>	

Syntax:	`Cal_ReturnType Cpl_<Primitive>(`
	`    <type> <xxx>`
	`)`	

Service ID[hex]:	--	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Cal_<Service>(), with the exception of the argument cfgId. This argument is of type â€œCal_ConfigIdTypeâ€œ in Cal_<Service>(). In Cpl_<Primitive> the argument cfgId shall be replaced by an argument cfgPtr of type â€œconst void *â€œ.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Cal_ReturnType	The return values shall be identical to those of the corresponding function Cal_<Service>().
Description:	This function shall process the cryptographic primitive with the given input data and store the result in the memory location given.	

⌋ ()

The API "Cpl_<Primitive>" has a parameter "cfgPtr" of type "const void *".
When calling this API, the parameter "cfgPtr" shall point to a constant variable of type "Cpl_<Primitive>ConfigType", but shall be cast to "const void *".
Reason for this is to have a common definition of the parameter list of this API for all primitives of one service, because in the structure Cal_<Service>ConfigType one element is a function pointer to this API.

8.4.3 Configuration of the cryptographic primitives

For each cryptographic primitive, a cryptographic library module has to provide a configuration structure. This configuration structure shall be of type

`Cpl_<Primitive>ConfigType`. For each configuration of a primitive, the cryptographic library module has to provide a constant variable of that type.
To link a primitive configuration to a specific service configuration, the corresponding parameter `Cal<Service>InitConfiguration` of the service configuration has to be set to the C-language symbol of the primitive configuration.

Variants of CPL modules with different optimization objectives may exist. These Variants should be handled by separate modules. Those optimizations may include execution speed, platform specific optimizations, RAM size and/or code segment size etc. The most suitable variant for a given deployment should be used.

9 Sequence diagrams

Not applicable.

10 Configuration

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.

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

Chapter 10.3 specifies published information of the module CAL.

The CAL library shall not have any configuration options that may affect the functional behaviour of the routines. I.e. for a given set of input parameters, the outputs shall be always the same. For example, the returned value in case of error shall not be configurable.
However, a library vendor is allowed to add specific configuration options concerning library implementation, e.g. for resources consumption optimization.

Note: When changing the configuration of a cryptographical service, the result of a routine may change even without changing the input parameters. This is no contradiction to SRS_LIBS_00001, because in this case a different configuration can be considered as using a different input parameter.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:
- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [4]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:
- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.2 Containers and configuration parameters

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

10.2.1 Variants

Variant1: This variant allows only pre-compile time configuration parameters.

10.2.2 Cal

Module Name	*Cal*
Module Description	Configuration of the Cal (CryptoAbstractionLibrary) module.
Post-Build Variant Support	false

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalAsymDecrypt	0..1	Container for incorporation of AsymDecrypt primitives.
CalAsymEncrypt	0..1	Container for incorporation of AsymEncrypt primitives.
CalAsymPrivateKeyExtract	0..1	Container for incorporation of AsymPrivateKeyExtract primitives.
CalAsymPrivateKeyWrapAsym	0..1	Container for incorporation of AsymPrivateKeyWrapAsym primitives.
CalAsymPrivateKeyWrapSym	0..1	Container for incorporation of AsymPrivateKeyWrapSym primitives.
CalAsymPublicKeyExtract	0..1	Container for incorporation of AsymPublicKeyExtract primitives.
CalChecksum	0..1	Container for incorporation of Checksum primitives.
CalCompression	0..1	Container for incorporation of Compression primitives.
CalDecompression	0..1	Container for incorporation of Decompression primitives.
CalGeneral	1	Container for common configuration options.
CalHash	0..1	Container for incorporation of Hash primitives.
CalKeyDerive	0..1	Container for incorporation of KeyDerive primitives.

CalKeyExchangeCalcPubVal	0..1	Container for incorporation of KeyExchangeCalcPubVal primitives.
CalKeyExchangeCalcSecret	0..1	Container for incorporation of KeyExchangeCalcSecret primitives.
CalMacGenerate	0..1	Container for incorporation of MacGenerate primitives.
CalMacVerify	0..1	Container for incorporation of MacVerify primitives.
CalRandomGenerate	0..1	Container for incorporation of RandomGenerate primitives.
CalRandomSeed	0..1	Container for incorporation of RandomSeed primitives.
CalSignatureGenerate	0..1	Container for incorporation of SignatureGenerate primitives
CalSignatureVerify	0..1	Container for incorporation of SignatureVerify primitives.
CalSymBlockDecrypt	0..1	Container for incorporation of SymBlockDecrypt primitives.
CalSymBlockEncrypt	0..1	Container for incorporation of SymBlockEncrypt primitives.
CalSymDecrypt	0..1	Container for incorporation of SymDecrypt primitives
CalSymEncrypt	0..1	Container for incorporation of SymEncrypt primitives.
CalSymKeyExtract	0..1	Container for incorporation of SymKeyExtract primitives.
CalSymKeyWrapAsym	0..1	Container for incorporation of SymKeyWrapAsym primitives.
CalSymKeyWrapSym	0..1	Container for incorporation of SymKeyWrapSym primitives.

10.2.3 CalGeneral

SWS Item	**ECUC_Cal_00554 :**
Container Name	CalGeneral
Description	Container for common configuration options.
Configuration Parameters	

SWS Item	**ECUC_Cal_00744 :**		
Name	CalMaxAlignScalarType		
Description	The scalar type which has the maximum alignment restrictions on the given platform. This type can be e.g. uint8, uint16 or uint32.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00799 :**		
Name	CalVersionInfoApi		
Description	Pre-processor switch to enable and disable availability of the API Cal_GetVersionInfo(). True: API Cal_GetVersionInfo() is available. False: API Cal_GetVersionInfo() is not available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	

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

No Included Containers

10.2.4 CalHash

SWS Item	**ECUC_Cal_00559 :**
Container Name	CalHash
Description	Container for incorporation of Hash primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00745 :**		
Name	CalHashMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a hash computation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalHashConfig	0..32	Configurations for the Hash service.

10.2.5 CalHashConfig

SWS Item	**ECUC_Cal_00560 :**
Container Name	CalHashConfig
Description	Configurations for the Hash service. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00563 :**
Name	CalHashInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1
Type	EcucStringParamDef
Default value	--
maxLength	--
minLength	--
regularExpression	--
Post-Build Variant Value	false

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

SWS Item	**ECUC_Cal_00562 :**		
Name	CalHashPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.6 CalMacGenerate

SWS Item	**ECUC_Cal_00635 :**
Container Name	CalMacGenerate
Description	Container for incorporation of MacGenerate primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00746 :**		
Name	CalMacGenerateMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a MAC generation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00709 :**	
Name	CalMacGenerateMaxKeySize	
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a MAC generation.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	--	
Post-Build Variant Value	false	

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

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalMacGenerateConfig	0..32	Configurations for the MacGenerate service.

10.2.7 CalMacGenerateConfig

SWS Item	**ECUC_Cal_00564 :**
Container Name	CalMacGenerateConfig
Description	Configurations for the MacGenerate service. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00567 :**		
Name	CalMacGenerateInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00566 :**		
Name	CalMacGeneratePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.8 CalMacVerify

SWS Item	**ECUC_Cal_00636 :**
Container Name	CalMacVerify
Description	Container for incorporation of MacVerify primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00747 :**		
Name	CalMacVerifyMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a MAC verification.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00710 :**		
Name	CalMacVerifyMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a MAC verification.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalMacVerifyConfig	0..32	Configurations for the MacVerify service.

10.2.9 CalMacVerifyConfig

SWS Item	**ECUC_Cal_00568 :**
Container Name	CalMacVerifyConfig
Description	Container for configuration of service MacVerify. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00571 :**
Name	CalMacVerifyInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1
Type	EcucStringParamDef

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

SWS Item	**ECUC_Cal_00570 :**		
Name	CalMacVerifyPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.10 CalRandomSeed

SWS Item	**ECUC_Cal_00641 :**
Container Name	CalRandomSeed
Description	Container for incorporation of RandomSeed primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00748 :**		
Name	CalRandomMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement seeding or generating a random number.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalRandomSeedConfig	0..32	Configurations for the RandomSeed service.

10.2.11 CalRandomSeedConfig

SWS Item	**ECUC_Cal_00642 :**
Container Name	CalRandomSeedConfig
Description	Container for configuration of service RandomSeed. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00645 :**		
Name	CalRandomSeedInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00644 :**		
Name	CalRandomSeedPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.12 CalRandomGenerate

SWS Item	**ECUC_Cal_00620 :**
Container Name	CalRandomGenerate
Description	Container for incorporation of RandomGenerate primitives.
Configuration Parameters	

Included Containers

Container Name	Multiplicity	Scope / Dependency
CalRandomGenerateConfig	0..32	Configurations for the RandomGenerate service.

10.2.13 CalRandomGenerateConfig

SWS Item	ECUC_Cal_00637 :
Container Name	CalRandomGenerateConfig
Description	Container for configuration of service RandomGenerate. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	ECUC_Cal_00640 :		
Name	CalRandomGenerateInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Cal_00639 :		
Name	CalRandomGeneratePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.14 CalSymBlockEncrypt

SWS Item	ECUC_Cal_00621 :
Container Name	CalSymBlockEncrypt

Description	Container for incorporation of SymBlockEncrypt primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00749 :**		
Name	CalSymBlockEncryptMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a symmetrical block encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00711 :**		
Name	CalSymBlockEncryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a symmetrical block encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSymBlockEncryptConfig	0..32	Configurations for the SymBlockEncrypt service.

10.2.15 CalSymBlockEncryptConfig

SWS Item	**ECUC_Cal_00572 :**
Container Name	CalSymBlockEncryptConfig
Description	Container for configuration of service SymBlockEncrypt. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00575 :**
Name	CalSymBlockEncryptInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1
Type	EcucStringParamDef
Default value	--
maxLength	--
minLength	--

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

SWS Item	**ECUC_Cal_00574 :**		
Name	CalSymBlockEncryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.16 CalSymBlockDecrypt

SWS Item	**ECUC_Cal_00622 :**
Container Name	CalSymBlockDecrypt
Description	Container for incorporation of SymBlockDecrypt primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00750 :**		
Name	CalSymBlockDecryptMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a symmetrical block decryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00712 :**	
Name	CalSymBlockDecryptMaxKeySize	
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a symmetrical block decryption.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	

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

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSymBlockDecryptConfig	0..32	Configurations for the SymBlockDecrypt service.

10.2.17 CalSymBlockDecryptConfig

SWS Item	**ECUC_Cal_00576 :**
Container Name	CalSymBlockDecryptConfig
Description	Container for configuration of service SymBlockDecrypt. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00579 :**		
Name	CalSymBlockDecryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00578 :**		
Name	CalSymBlockDecryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.18 CalSymEncrypt

SWS Item	**ECUC_Cal_00623 :**
Container Name	CalSymEncrypt
Description	Container for incorporation of SymEncrypt primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00751 :**		
Name	CalSymEncryptMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a symmetrical encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00713 :**		
Name	CalSymEncryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a symmetrical encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSymEncryptConfig	0..32	Configurations for the SymEncrypt service.

10.2.19 CalSymEncryptConfig

SWS Item	**ECUC_Cal_00580 :**
Container Name	CalSymEncryptConfig
Description	Container for configuration of service SymEncrypt. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00583 :**
Name	CalSymEncryptInitConfiguration

Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00582 :**		
Name	CalSymEncryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.20 CalSymDecrypt

SWS Item	**ECUC_Cal_00624 :**
Container Name	CalSymDecrypt
Description	Container for incorporation of SymDecrypt primitives
Configuration Parameters	

SWS Item	**ECUC_Cal_00752 :**		
Name	CalSymDecryptMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a symmetrical decryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00714 :**		
Name	CalSymDecryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a symmetrical decryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
CalSymDecryptConfig	0..32	Configurations for the SymDecrypt service.

10.2.21 CalSymDecryptConfig

SWS Item	**ECUC_Cal_00584 :**
Container Name	CalSymDecryptConfig
Description	Container for configuration of service SymDecrypt. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00587 :**		
Name	CalSymDecryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00586 :**		
Name	CalSymDecryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants

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

No Included Containers

10.2.22 CalAsymEncrypt

SWS Item	**ECUC_Cal_00625 :**
Container Name	CalAsymEncrypt
Description	Container for incorporation of AsymEncrypt primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00753 :**		
Name	CalAsymEncryptMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement an asymmetrical encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00715 :**		
Name	CalAsymEncryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement an asymmetrical encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalAsymEncryptConfig	0..32	Configurations for the AsymEncrypt service.

10.2.23 CalAsymEncryptConfig

SWS Item	**ECUC_Cal_00588 :**
Container Name	CalAsymEncryptConfig

Description	Container for configuration of service AsymEncrypt. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00591 :**		
Name	CalAsymEncryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00590 :**		
Name	CalAsymEncryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.24 CalAsymDecrypt

SWS Item	**ECUC_Cal_00626 :**
Container Name	CalAsymDecrypt
Description	Container for incorporation of AsymDecrypt primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00754 :**	
Name	CalAsymDecryptMaxCtxBufByteSize	
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement an asymmetrical decryption.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	--	
Post-Build Variant Value	false	

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

SWS Item	**ECUC_Cal_00716 :**		
Name	CalAsymDecryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement an asymmetrical decryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalAsymDecryptConfig	0..32	Configurations for the AsymDecrypt service.

10.2.25 CalAsymDecryptConfig

SWS Item	**ECUC_Cal_00592 :**
Container Name	CalAsymDecryptConfig
Description	Container for configuration of service AsymDecrypt. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00595 :**		
Name	CalAsymDecryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00594 :**
Name	CalAsymDecryptPrimitiveName
Description	Name of the cryptographic primitive to use.
Multiplicity	1
Type	EcucStringParamDef
Default value	--

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

No Included Containers

10.2.26 CalSignatureGenerate

SWS Item	**ECUC_Cal_00627 :**
Container Name	CalSignatureGenerate
Description	Container for incorporation of SignatureGenerate primitives
Configuration Parameters	

SWS Item	**ECUC_Cal_00755 :**		
Name	CalSignatureGenerateMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a signature generation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00717 :**		
Name	CalSignatureGenerateMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a signature generation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSignatureGenerateConfig	0..32	Configurations for the SignatureGenerate service.

10.2.27 CalSignatureGenerateConfig

SWS Item	**ECUC_Cal_00596 :**
Container Name	CalSignatureGenerateConfig
Description	Container for configuration of service SignatureGenerate. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00599 :**		
Name	CalSignatureGenerateInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00598 :**		
Name	CalSignatureGeneratePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.28 CalSignatureVerify

SWS Item	**ECUC_Cal_00628 :**
Container Name	CalSignatureVerify
Description	Container for incorporation of SignatureVerify primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00756 :**
Name	CalSignatureVerifyMaxCtxBufByteSize
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a signature verification.

Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00718 :**		
Name	CalSignatureVerifyMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a signature verification.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSignatureVerifyConfig	0..32	Configurations for the SignatureVerify service.

10.2.29 CalSignatureVerifyConfig

SWS Item	**ECUC_Cal_00600 :**
Container Name	CalSignatureVerifyConfig
Description	Container for configuration of service SignatureVerify. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00603 :**		
Name	CalSignatureVerifyInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00602 :**

Name	CalSignatureVerifyPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.30 CalCompression

SWS Item	**ECUC_Cal_00789 :**
Container Name	CalCompression
Description	Container for incorporation of Compression primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00790 :**		
Name	CalCompressMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a compression computation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalCompressionConfig	0..32	Container for configuration of service Compression. The container name serves as a symbolic name for the identifier of a service configuration.

10.2.31 CalCompressionConfig

SWS Item	**ECUC_Cal_00791 :**
Container Name	CalCompressionConfig
Description	Container for configuration of service Compression. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00792 :**		
Name	CalCompressInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00793 :**		
Name	CalCompressPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.32 CalDecompression

SWS Item	**ECUC_Cal_00794 :**
Container Name	CalDecompression
Description	Container for incorporation of Decompression primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00795 :**		
Name	CalDecompressMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a decompression computation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalDecompressionConfig	0..32	Container for configuration of service Decompression. The container name serves as a symbolic name for the identifier of a service configuration.

10.2.33 CalDecompressionConfig

SWS Item	**ECUC_Cal_00796 :**
Container Name	CalDecompressionConfig
Description	Container for configuration of service Decompression. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00797 :**		
Name	CalDecompressInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00798 :**		
Name	CalDecompressPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.34 CalChecksum

SWS Item	**ECUC_Cal_00629 :**		
Container Name	CalChecksum		
Description	Container for incorporation of Checksum primitives.		
Configuration Parameters			

SWS Item	**ECUC_Cal_00757 :**		
Name	CalChecksumMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a checksum computation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalChecksumConfig	0..32	Configurations for the Checksum service.

10.2.35 CalChecksumConfig

SWS Item	**ECUC_Cal_00604 :**
Container Name	CalChecksumConfig
Description	Container for configuration of service Checksum. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00607 :**		
Name	CalChecksumInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00606 :**
Name	CalChecksumPrimitiveName
Description	Name of the cryptographic primitive to use.

Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.36 CalKeyDerive

SWS Item	**ECUC_Cal_00630 :**
Container Name	CalKeyDerive
Description	Container for incorporation of KeyDerive primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00758 :**		
Name	CalKeyDeriveMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a key derivation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00719 :**		
Name	CalKeyDeriveMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRL primitives which implement a key derivation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalKeyDeriveConfig	0..32	Configurations for the KeyDerive service.

10.2.37 CalKeyDeriveConfig

SWS Item	**ECUC_Cal_00608 :**
Container Name	CalKeyDeriveConfig
Description	Container for configuration of service KeyDerive. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00611 :**		
Name	CalKeyDeriveInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00610 :**		
Name	CalKeyDerivePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.38 CalKeyExchangeCalcPubVal

SWS Item	**ECUC_Cal_00631 :**
Container Name	CalKeyExchangeCalcPubVal
Description	Container for incorporation of KeyExchangeCalcPubVal primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00720 :**

Name	CalKeyExchangeCalcPubValMaxBaseTypeSize		
Description	The maximum length, in bytes, of all base types used in all CPL primitives which implement a public value calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00759 :**		
Name	CalKeyExchangeCalcPubValMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a public value calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00721 :**		
Name	CalKeyExchangeCalcPubValMaxPrivateTypeSize		
Description	The maximum length, in bytes, of all private information types used in all CPL primitives which implement a public value calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalKeyExchangeCalcPubValConfig	0..32	Configurations for the KeyExchangeCalcPubVal

10.2.39 CalKeyExchangeCalcPubValConfig

SWS Item	**ECUC_Cal_00612 :**
Container Name	CalKeyExchangeCalcPubValConfig
Description	Container for configuration of service KeyExchangeCalcPubVal. The container name serves as a symbolic name for the identifier of a service configuration.

Configuration Parameters

SWS Item	**ECUC_Cal_00615 :**		
Name	CalKeyExchangeCalcPubValInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00614 :**		
Name	CalKeyExchangeCalcPubValPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.40 CalKeyExchangeCalcSecret

SWS Item	**ECUC_Cal_00632 :**
Container Name	CalKeyExchangeCalcSecret
Description	Container for incorporation of KeyExchangeCalcSecret primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00722 :**		
Name	CalKeyExchangeCalcSecretMaxBaseTypeSize		
Description	The maximum length, in bytes, of all base types used in all CPL primitives which implement a shared secret calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants

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

SWS Item	**ECUC_Cal_00760 :**		
Name	CalKeyExchangeCalcSecretMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a shared secret calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00723 :**		
Name	CalKeyExchangeCalcSecretMaxPrivateTypeSize		
Description	The maximum length, in bytes, of all private information types used in all CPL primitives which implement a shared secret calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalKeyExchangeCalcSecretConfig	0..32	Configurations for the KeyExchangeCalcSecret service.

10.2.41 CalKeyExchangeCalcSecretConfig

SWS Item	**ECUC_Cal_00616 :**
Container Name	CalKeyExchangeCalcSecretConfig
Description	Container for configuration of service KeyExchangeCalcSecret. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00545 :**
Name	CalKeyExchangeCalcSecretInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1
Type	EcucStringParamDef
Default value	--

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

SWS Item	**ECUC_Cal_00618 :**		
Name	CalKeyExchangeCalcSecretPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.42 CalSymKeyExtract

SWS Item	**ECUC_Cal_00633 :**
Container Name	CalSymKeyExtract
Description	Container for incorporation of SymKeyExtract primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00761 :**		
Name	CalSymKeyExtractMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a symmetrical key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00724 :**
Name	CalSymKeyExtractMaxKeySize
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a symmetrical key extraction.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSymKeyExtractConfig	0..32	Configurations for the SymKeyExtract service.

10.2.43 CalSymKeyExtractConfig

SWS Item	**ECUC_Cal_00546 :**
Container Name	CalSymKeyExtractConfig
Description	Container for configuration of service SymKeyExtract. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00549 :**		
Name	CalSymKeyExtractInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00548 :**		
Name	CalSymKeyExtractPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.44 CalAsymPublicKeyExtract

SWS Item	**ECUC_Cal_00634 :**
Container Name	CalAsymPublicKeyExtract
Description	Container for incorporation of AsymPublicKeyExtract primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00762 :**		
Name	CalAsymPublicKeyExtractMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement an asymmetrical public key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00725 :**		
Name	CalAsymPublicKeyExtractMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement an asymmetrical public key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalAsymPublicKeyExtractConfig	0..32	Configurations for the AsymPublicKeyExtract service.

10.2.45 CalAsymPublicKeyExtractConfig

SWS Item	**ECUC_Cal_00550 :**
Container Name	CalAsymPublicKeyExtractConfig
Description	Container for configuration of service AsymPublicKeyExtract. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00553 :**		
Name	CalAsymPublicKeyExtractInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00552 :**		
Name	CalAsymPublicKeyExtractPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.46 CalAsymPrivateKeyExtract

SWS Item	**ECUC_Cal_00686 :**
Container Name	CalAsymPrivateKeyExtract
Description	Container for incorporation of AsymPrivateKeyExtract primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00763 :**		
Name	CalAsymPrivateKeyExtractMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement an asymmetrical private key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00726 :**		
Name	CalAsymPrivateKeyExtractMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement an asymmetrical private key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalAsymPrivateKeyExtractConfig	0..32	Configurations for the AsymPrivateKeyExtract.

10.2.47 CalAsymPrivateKeyExtractConfig

SWS Item	**ECUC_Cal_00687 :**
Container Name	CalAsymPrivateKeyExtractConfig
Description	Container for configuration of service AsymPrivateKeyExtract. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00690 :**		
Name	CalAsymPrivateKeyExtractInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00689 :**
Name	CalAsymPrivateKeyExtractPrimitiveName
Description	Name of the cryptographic primitive to use.
Multiplicity	1
Type	EcucStringParamDef
Default value	--
maxLength	--
minLength	--

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

No Included Containers

10.2.48 CalSymKeyWrapAsym

SWS Item	**ECUC_Cal_00765 :**
Container Name	CalSymKeyWrapAsym
Description	Container for incorporation of SymKeyWrapAsym primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00800 :**		
Name	CalSymKeyWrapAsymMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a asymmetrical wrapping of a symmetric key.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00785 :**		
Name	CalSymKeyWrapAsymMaxPubKeySize		
Description	The maximum length, in bytes, of all public key types used in all CPL primitives which implement a symmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00786 :**
Name	CalSymKeyWrapAsymMaxSymKeySize
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a symmetrical key wrapping.
Multiplicity	1
Type	EcucIntegerParamDef
Range	1 .. 4294967295
Default value	--
Post-Build Variant Value	false

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

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSymKeyWrapAsymConfig	0..32	Container for configuration of service SymKeyWrapAsym. The container name serves as a symbolic name for the identifier of a service configuration.

10.2.49 CalSymKeyWrapAsymConfig

SWS Item	**ECUC_Cal_00782 :**
Container Name	CalSymKeyWrapAsymConfig
Description	Container for configuration of service SymKeyWrapAsym. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00784 :**		
Name	CalSymKeyWrapAsymInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00783 :**		
Name	CalSymKeyWrapAsymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.50 CalSymKeyWrapSym

SWS Item	**ECUC_Cal_00764 :**
Container Name	CalSymKeyWrapSym
Description	Container for incorporation of SymKeyWrapSym primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00801 :**		
Name	CalSymKeyWrapSymMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a symmetrical wrapping of a symmetric key.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00781 :**		
Name	CalSymKeyWrapSymMaxSymKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement a symmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalSymKeyWrapSymConfig	0..32	Container for configuration of service SymKeyWrapSym. The container name serves as a symbolic name for the identifier of a service configuration.

10.2.51 CalSymKeyWrapSymConfig

SWS Item	**ECUC_Cal_00777 :**
Container Name	CalSymKeyWrapSymConfig
Description	Container for configuration of service SymKeyWrapSym. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00779 :**		
Name	CalSymKeyWrapSymInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00778 :**		
Name	CalSymKeyWrapSymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.52 CalAsymPrivateKeyWrapAsym

SWS Item	**ECUC_Cal_00767 :**
Container Name	CalAsymPrivateKeyWrapAsym
Description	Container for incorporation of AsymPrivateKeyWrapAsym primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00802 :**		
Name	CalAsymPrivateKeyWrapAsymMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a asymmetrical wrapping of the private part of an asymmetric key.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

SWS Item	**ECUC_Cal_00771 :**		
Name	CalAsymPrivateKeyWrapAsymMaxPrivKeySize		
Description	The maximum length, in bytes, of all private information types used in all CPL primitives which implement an asymmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00787 :**		
Name	CalAsymPrivateKeyWrapAsymMaxPubKeySize		
Description	The maximum length, in bytes, of all public key types used in all CPL primitives which implement an asymmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalAsymPrivateKeyWrapAsymConfig	0..32	Container for configuration of service AsymPrivateKeyWrapAsym. The container name serves as a symbolic name for the identifier of a service configuration.

10.2.53 CalAsymPrivateKeyWrapAsymConfig

SWS Item	**ECUC_Cal_00768 :**
Container Name	CalAsymPrivateKeyWrapAsymConfig
Description	Container for configuration of service AsymPrivateKeyWrapAsym. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00770 :**
Name	CalAsymPrivateKeyWrapAsymInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.

Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00769 :**		
Name	CalAsymPrivateKeyWrapAsymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.54 CalAsymPrivateKeyWrapSym

SWS Item	**ECUC_Cal_00766 :**
Container Name	CalAsymPrivateKeyWrapSym
Description	Container for incorporation of AsymPrivateKeyWrapSym primitives.
Configuration Parameters	

SWS Item	**ECUC_Cal_00803 :**		
Name	CalAsymPrivateKeyWrapSymMaxCtxBufByteSize		
Description	The maximum, in bytes, of all context buffers used in all CPL primitives which implement a symmetrical wrapping of the private part of an asymmetric key.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00775 :**		
Name	CalAsymPrivateKeyWrapSymMaxPrivKeySize		
Description	The maximum length, in bytes, of all private information types used in all CPL primitives which implement an asymmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Cal_00776 :**		
Name	CalAsymPrivateKeyWrapSymMaxSymKeySize		
Description	The maximum, in bytes, of all key lengths used in all CPL primitives which implement an asymmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CalAsymPrivateKeyWrapSymConfig	0..32	Container for configuration of service AsymPrivateKeyWrapSym. The container name serves as a symbolic name for the identifier of a service configuration.

10.2.55 CalAsymPrivateKeyWrapSymConfig

SWS Item	**ECUC_Cal_00772 :**
Container Name	CalAsymPrivateKeyWrapSymConfig
Description	Container for configuration of service AsymPrivateKeyWrapSym. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**ECUC_Cal_00774 :**
Name	CalAsymPrivateKeyWrapSymInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1
Type	EcucStringParamDef
Default value	--
maxLength	--

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

SWS Item	**ECUC_Cal_00773 :**		
Name	CalAsymPrivateKeyWrapSymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.3 Published Information

[SWS_Cal_00780]⌈ The standardized common published parameters as required by SRS_BSW_00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋ (SRS_BSW_00402, SRS_BSW_00003)

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

11 Not applicable requirements

[SWS_Cal_00781]⌈ These input requirements are not applicable to this specification.⌋(SRS_BSW_00411, SRS_BSW_00101, SRS_BSW_00164, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00314, SRS_BSW_00358, SRS_BSW_00467)