Document Title	Specification of Crypto Service Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	402
Document Classification	Standard

Document Version	1.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
09.12.2011	1.2.0	AUTOSAR Administration	• Fixed issues with AUTOSAR Port Interfaces
24.11.2010	1.1.0	AUTOSAR Administration	• Complete Configuration parameters • Complete API specifications • Add support for secure key storage • Integration of support for key transport services • Introduction of new DET error (checking of the null pointer in getversion info).
30.11.2009	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the software module Crypto Service Manager (CSM) to satisfy the top-level requirements represented in the CSM Requirements Specification (SRS) [CSM_SRS].

The CSM shall provide synchronous or asynchronous services to enable a unique access to basic cryptographic functionalities for all software modules. The CSM shall provide an abstraction layer, which offers a standardized interface to higher software layers to access these functionalities.

The functionality required by a software module can be different to the functionality required by other software modules. For this reason there shall be the possibility to configure and initialize the services provided by the CSM individually for each software module. This configuration comprises as well the selection of synchronous or asynchronous processing of the CSM services.

The construction of the CSM module follows a generic approach. Wherever a detailed specification of structures and interfaces would limit the scope of the usability of the CSM, 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 [13], are listed in this chapter.

Abbreviation / Acronym:	***Description:***
DEM / Dem	Diagnostic Event Manager
DET / Det	Development Error Tracer
CSM / Csm	Crypto Service Manager
CRY / Cry	Cryptographic library module

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[5] Specification of BSW Scheduler
AUTOSAR_SWS_Scheduler.pdf

[6] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[7] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping.pdf

[8] Specification of Development Error Tracer
UTOSAR_SWS_DevelopmentErrorTracer.pdf

[9] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[10] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[11] Specification of C Implementation Rules
AUTOSAR_TR_CImplementationRules.pdf

[12] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[13] AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf

[14] Requirements on Crypto Service Manager
AUTOSAR_SRS_CryptoServiceManager.pdf

[15] Specification of Crypto Abstraction Library
AUTOSAR_SWS_CryptoAbstractionLibrary.pdf

3.2 Related standards and norms

[5] 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.

4.3 Security implications

The AUTOSAR Service implementation of the CSM makes it possible for software components to use the services of the CSM via the Virtual Function Bus (VFB). In this respect the CSM and a software component can either be located on the same ECU, or they can be located on different ECUs.
If the CSM and the SWC are located on different ECUs, it should be taken into account that this might raise some security implications:
Communication between ECUs is done e.g. via CAN bus. This means, that the unencrypted information, that shall be encrypted by the CSM, has to be transmitted between the ECUs before encryption. Also key material has to be transmitted via the CAN bus.

5 Dependencies to other modules

CSM0001

The CSM shall be able to incorporate cryptographic library modules, which are implemented according to the cryptographic library requirement specification in chapter 8.4.

CSM0506

The CSM module 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.

CSM0528

The CSM module is using services of the DET module for tracing development errors.

CSM0529

The CSM module is using services of the DEM module for tracing production errors.

5.1 File structure

5.1.1 Code file structure

CSM0002

The code file structure shall not be defined within this specification completely. The CSM module shall consist of the following parts:

CSM0006

The code file structure shall contain one or more MISRA-C 2004 conform source files Csm_<xxx>.c, that contain the entire parts of the CSM code.

CSM0692

The code file structure shall contain one or more MISRA-C 2004 conform source files Cry_<xxx>.c, that contain the entire code of the incorporated cryptographic library modules.

5.1.2 Header file structure

CSM0693

The header file structure shall not be defined within this specification completely The CSM module shall provide the following headers:

CSM0005

The header file structure shall contain an application interface header file Csm.h, that provides the function prototypes to access the CSM services.

CSM0003

The header file structure shall contain a configuration header Csm_Cfg.h, that provides the configuration parameters for the CSM module.

CSM0727

The header file structure shall contain a callback interface Csm_Cbk.h, that provides the callback function prototypes to be used by the underlying cryptographic library modules.

CSM0677

The header file structure shall contain an AUTOSAR service interface header Rte_Csm.h, that provides the interfaces to the RTE. This file is generated by the RTE.

CSM0678

The header file structure shall contain a type header Rte_Csm_Type.h that contains all types, that are used by both AUTOSAR interfaces and BSW module interfaces. This file is generated by the RTE.

CSM0004

The header file structure shall contain a type header Csm_Types.h, that provides the types, particularly configuration types, for the CSM module. This file shall only contain types, that are not already defined in Rte_Csm_Type.h

CSM0694

Each underlying cryptographic library module shall provide a header file Cry_<xxx>.h.

CSM0008

The Figure in CSM0695 (CSM File Structure) shows the include file structure, which shall be as follows:

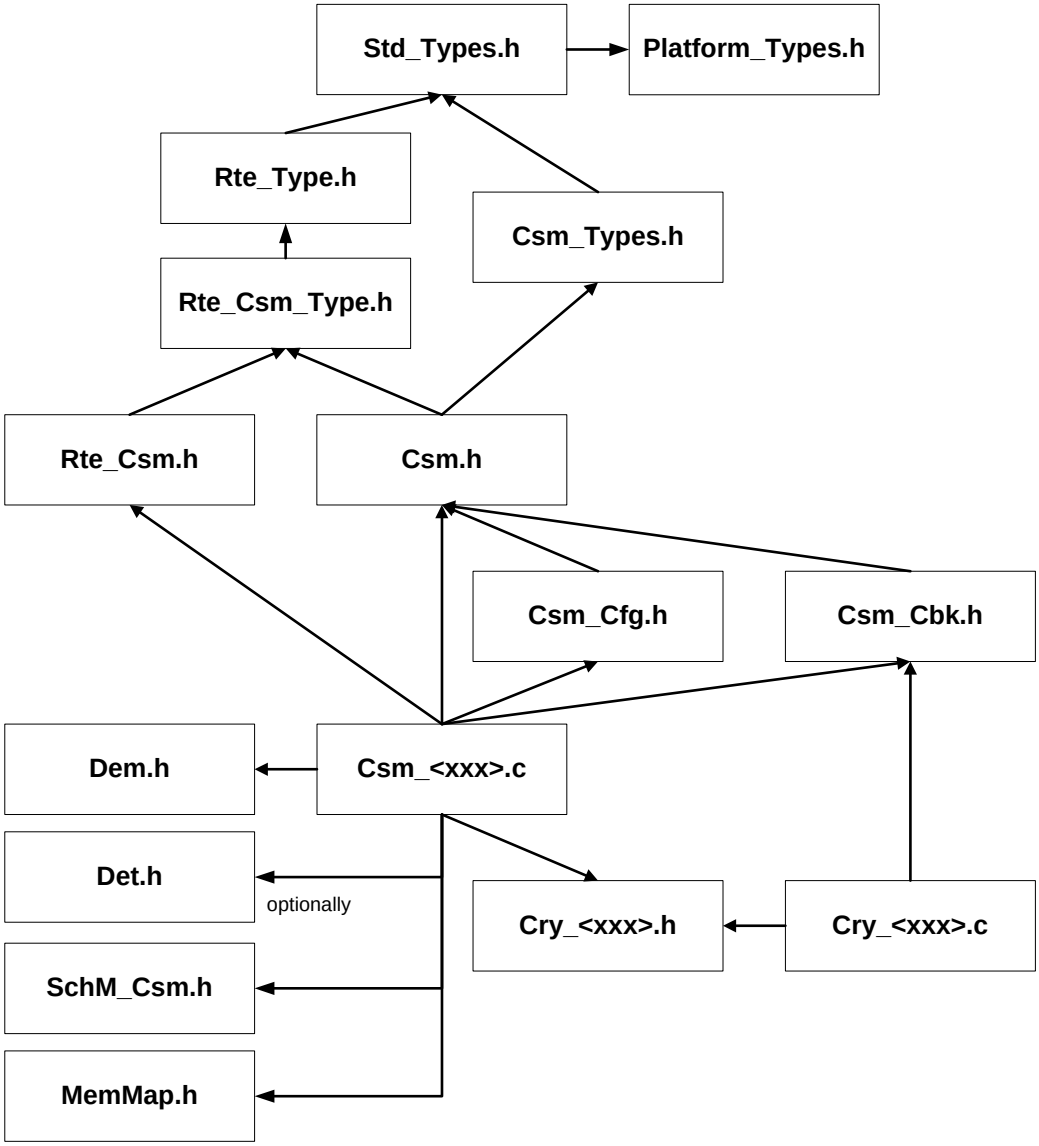
- Csm_Cfg.h shall include Csm.h.
- Csm_Types.h shall include Std_Types.h.
- Csm.h shall include Csm_Types.h and Rte_Csm_Type.h.

- Csm_<xxx>.c shall include Csm.h, Rte_Csm.h, Csm_Cfg.h, Csm_Cbk.h, Dem.h, SchM_Csm.h, MemMap.h and optionally Det.h if the development error detection is turned on.

- Csm_<xxx>.c shall include Cry_<xxx>.h

- Cry_<xxx>.c shall include Cry_<xxx>.h and Csm_Cbk.h

CSM0695



CSM0009

The CSM module shall include the Dem.h file. This inclusion allows the CSM to report errors with the required Event Id symbols.

6 Requirements traceability

Document: AUTOSAR requirements on Basic Software, general

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	Not applicable (CSM is only pre-compile-time configurable)
[BSW00404] Reference to post build time configuration	Not applicable (CSM is only pre-compile-time configurable)
[BSW00405] Reference to multiple configuration sets	Not applicable (CSM is only pre-compile-time configurable)
[BSW00345] Pre-compile-time configuration	CSM0003
[BSW159] Tool-based configuration	Chapter 10
[BSW167] Static configuration checking	Chapter 10, CSM0030
[BSW171] Configurability of optional functionality	Chapter 10, CSM0015
[BSW170] Data for reconfiguration of AUTOSAR SW-components	Not applicable (CSM is no AUTOSAR SW-C)
[BSW00380] Separate C-File for configuration parameters	Not applicable (CSM is only pre-compile-time configurable)
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Chapter 5.1.1
[BSW00381] Separate configuration header file for pre-compile time parameters	Chapter 5.1.2
[BSW00412] Separate H-File for configuration parameters	Not applicable (CSM is only pre-compile-time configurable)
[BSW00383] List dependencies of configuration files	Chapter 5
[BSW00384] List dependencies to other module	Chapter 5
[BSW00387] Specify the configuration class of callback functions	CSM0073, Chapter 10.2
[BSW00388] Introduce containers	Chapter 10
[BSW00389] Containers shall have names	Chapter 10
[BSW00390] Parameter content shall be unique within the module	Chapter 10
[BSW00391] Parameter shall have unique names	Chapter 10
[BSW00392] Parameters shall have a type	Chapter 10

[BSW00393] Parameters shall have a range	Chapter 10
[BSW00394] Specify the scope of the parameters	Chapter 10
[BSW00395] List the required parameters (per parameter]	Chapter 10
[BSW00396] Configuration classes	Chapter 10
[BSW00397] Pre-compile-time parameters	Chapter 10
[BSW00398] Link-time-parameters	Not applicable (CSM is only pre-compile-time configurable)
[BSW00399] Loadable post-build time parameters	Not applicable (CSM is only pre-compile-time configurable)
[BSW00400] Selectable post-build time parameters	Not applicable (CSM is only pre-compile-time configurable)
[BSW00438] Post Build Configuration Data Structure	Not applicable (CSM is only pre-compile-time configurable)
[BSW00402] Published information	Chapter 10.3, CSM0504
[BSW00375] Notification of wake-up reason	Not applicable (no use case for the CSM module)
[BSW101] Initialization interface	CSM0646
[BSW00416] Sequence of Initialization	Not applicable (CSM is not responsible for any BSW module initialization)
[BSW00406] Check module initialization	CSM0539, CSM0064
[BSW00437] NoInit--Area in RAM	Not applicable (no use case for the CSM module)
[BSW168] Diagnostic Interface	Not applicable (no use case for the CSM module)
[BSW00407] Function to read out published parameters	CSM0705
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Chapter 11
[BSW00424] BSW main processing function task allocation	Not applicable (implementation requirement)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (no use case for the CSM module)
[BSW00426] Exclusive areas in BSW modules	Not applicable (no use case for the CSM module)
[BSW00427] ISR description for BSW modules	Not applicable (no use case for the CSM module)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (no use case for the CSM module)

[BSW00429] Restricted BSW OS functionality access	Not applicable (no use case for the CSM module)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (implementation requirement)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (no use case for the CSM module)
[BSW00433] Calling of main processing functions	Chapter 8.6, CSM0476
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (implementation requirement)
[BSW00336] Shutdown interface	Not applicable (no use case for the CSM module)
[BSW00337] Classification of errors	CSM0064, CSM0062, CSM0555_Conf
[BSW00338] Detection and reporting of development errors	CSM0067, CSM0062, CSM0555_Conf
[BSW00369] Do not return development error codes via API	CSM0488
[BSW00339] Reporting of production relevant errors status	CSM0065, CSM0066
[BSW00422] Pre--de--bouncing of production relevant error status	Not applicable (requirement on DEM)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (CSM is part of Basic Software)
[BSW00323] API parameter checking	CSM0063, CSM0062, CSM0555_Conf
[BSW004] Version check	CSM0060
[BSW00409] Header files for production code error IDs	CSM0538
[BSW00385] List possible error notifications	CSM0064, CSM0539
[BSW00386] Configuration for detecting an error	CSM0062, CSM0065
[BSW161] Microcontroller abstraction	Not applicable (requirement on AUTOSAR architecture)
[BSW162] ECU layout abstraction	Not applicable (requirement on AUTOSAR architecture)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on AUTOSAR architecture)
[BSW00415] User dependent include files	Not applicable (no use case for the CSM module)
[BSW164] Implementation of interrupt service routines	Not applicable (CSM does not use any ISRs)
[BSW00325] Runtime of interrupt service routines	Not applicable (CSM does not use any ISRs)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (CSM does not use any ISRs)
[BSW00342] Usage of source code and object code	Not applicable (requirement on AUTOSAR architecture)
[BSW00343] Specification and	CSM0558_Conf

configuration of time	
[BSW160] Human-readable configuration data	Not applicable (implementation requirement)
[BSW007] HIS MISRA C	CSM0006, CSM0692
[BSW00300] Module naming convention	Chapter 5.1
[BSW00413] Accessing instances of BSW modules	Not applicable (no use case for the CSM module)
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (CSM is no driver module)
[BSW00441] Enumeration literals and #define naming convention	Chapter 8.2, chapter 8.4.1
[BSW00305] Self-defined data types naming convention	Chapter 8.2, chapter 8.4.1
[BSW00307] Global variables naming convention	Not applicable (implementation requirement)
[BSW00310] API naming convention	Chapters 8.3, 8.4.2, 8.5, 8.6
[BSW00373] Main processing function naming convention	CSM0479
[BSW00327] Error values naming convention	CSM0069, CSM0679
[BSW00335] Status values naming convention	Not applicable (implementation requirement)
[BSW00350] Development error detection keyword	CSM0062
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	Not applicable (implementation requirement)
[BSW00411] Get version info keyword	CSM0707
[BSW00346] Basic set of module files	Chapter 5.1
[BSW158] Separation of configuration from implementation	Chapter 5.1
[BSW00314] Separation of interrupt frames and service routines	Not applicable (no use case for the CSM module)
[BSW00370] Separation of callback interface from API	CSM0727
[BSW00435] Header File Structure for the Basic Software Scheduler	CSM0008
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	CSM0008
[BSW00348] Standard type header	CSM0008
[BSW00353] Platform specific type header	Not applicable (CSM includes standard type header)
[BSW00361] Compiler specific language extension header	Not applicable (CSM includes standard type header)
[BSW00301] Limit imported information	Not applicable (implementation requirement)
[BSW00302] Limit exported information	Not applicable

	(implementation requirement)
[BSW00328] Avoid duplication of code	Not applicable (implementation requirement)
[BSW00312] Shared code shall be reentrant	Not applicable (implementation requirement)
[BSW006] Platform independency	Not applicable (implementation requirement)
[BSW00439] Declaration of interrupt handlers and ISRs	Not applicable (CSM does not use interrupts and ISRs)
[BSW00357] Standard API return type	Chapter 8.2
[BSW00377] Module specific API return types	Chapters 8.2, 8.4.1
[BSW00304] AUTOSAR integer data types	Chapter 8
[BSW00355] Do not redefine AUTOSAR integer data types	Chapter 8
[BSW00378] AUTOSAR boolean type	Chapter 8
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (implementation requirement)
[BSW00308] Definition of global data	Not applicable (implementation requirement)
[BSW00309] Global data with read-only constraint	Not applicable (implementation requirement)
[BSW00371] Do not pass function pointers via API	Chapter 8.3
[BSW00358] Return type of init() functions	CSM0646
[BSW00414] Parameter of init function	CSM0646
[BSW00376] Return type and parameters of main processing functions	CSM0479
[BSW00359] Return type of callback functions	CSM0073, CSM0455, CSM0457
[BSW00360] Parameters of callback functions	CSM0073, CSM0455, CSM0457
[BSW00440] Function prototype for callback functions of AUTOSAR Services	Not applicable (names of function prototypes for callback functions have to be configured)
[BSW00329] Avoidance of generic interfaces	Chapter 8.3
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (implementation requirement)
[BSW00331] Separation of error and status values	Not applicable (implementation requirement)
[BSW009] Module User Documentation	Not applicable (requirement on documentation)
[BSW00401] Documentation of multiple instances of configuration parameters	Chapter 10
[BSW172] Compatibility and documentation of scheduling strategy	Chapter 8.6

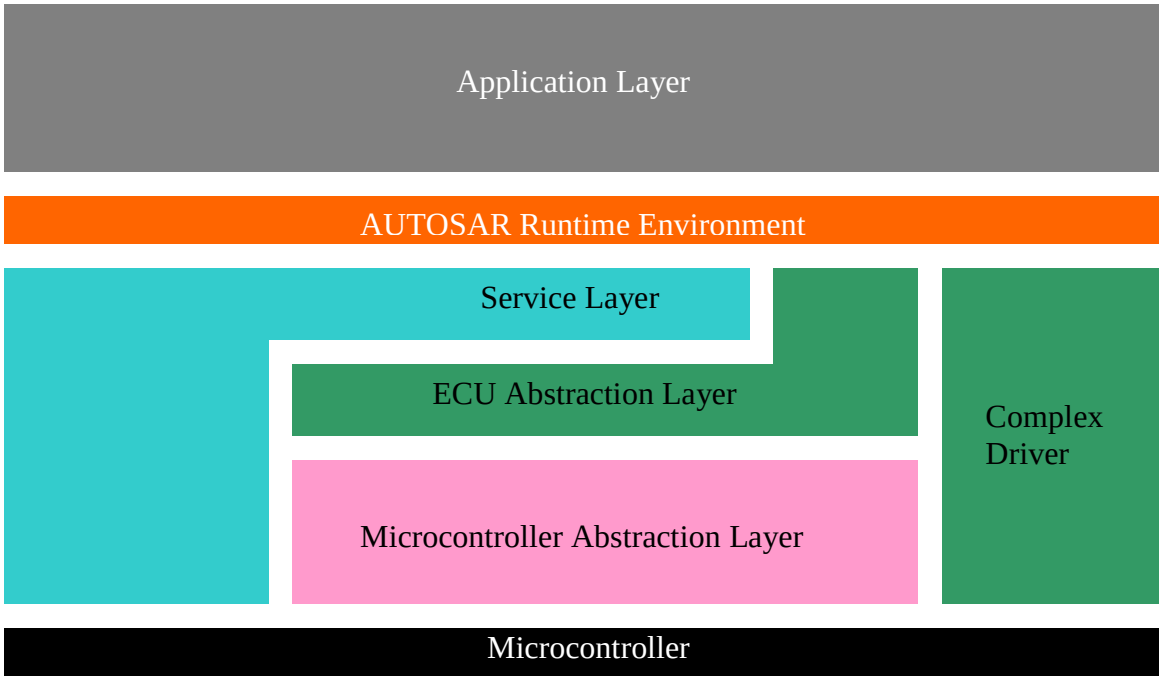
[BSW010] Memory resource documentation	Not applicable (requirement on documentation)
[BSW00333] Documentation of callback function context	Not applicable (requirement on documentation)
[BSW00374] Module vendor identification	CSM0504
[BSW00379] Module identification	CSM0504
[BSW003] Version identification	CSM0504
[BSW00318] Format of module version numbers	CSM0504
[BSW00321] Enumeration of module version numbers	Not applicable (implementation requirement)
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on documentation)
[BSW00334] Provision of XML file	Not applicable (requirement on documentation)

Document: Requirements on CSM

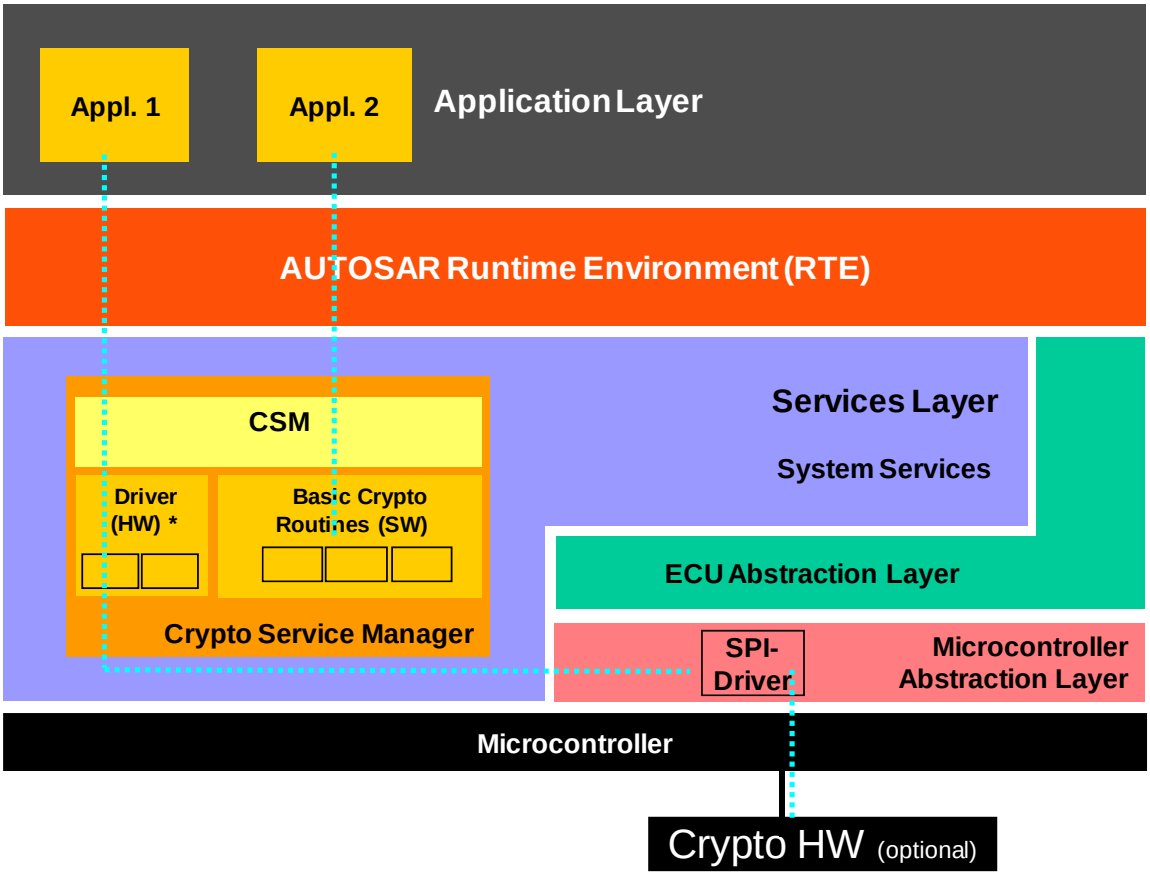
Requirement	***Satisfied by***
[BSW42600061] General interfaces	Chapter 7.2.2, Chapter 10.2.3
[BSW42600001] scalability	CSM0015
[BSW42600002] CRY interface	Chapter 8.4
[BSW42600069] CRY interface specification	Chapter 8.4
[BSW42600010] role of cryptographic primitives	Chapter 8.4
[BSW42600011] internal CRY interface	Chapter 8.4
[BSW42600004] configuration rules	CSM0030
[BSW42600005] job processing mode	CSM0557_Conf; CSM0505
[BSW42600006] cryptographic services	CSM0461
[BSW42600007] other modules	Chapter 5
[BSW42600008] callback function	CSM0032
[BSW42600009] initialization function	CSM0021
[BSW42600030] streaming approach	CSM0023
[BSW42600063] streaming approach	Chapter 8.4
[BSW42600012] error types	Chapter 7.4 and 7.5
[BSW42600013] development errors	Chapter 7.4 and 7.5
[BSW42600014] development error codes via API	Chapter 7.5
[BSW42600015] parameter checking	Chapter 8.7
[BSW42600047] abstraction layer	Chapter 1
[BSW42600064] location in service layer	Chapter 7
[BSW42600068] RTE interface for	Chapter 11

callbacks	
[BSW42600067] RTE interface for services	Chapter 11
[BSW42600066] general RTE interface	Chapter 11
[BSW42600060] configuration files	Chapter 5.1
[BSW42600036] implementation	CSM0006
[BSW42600046] error and status information	CSM0069
[BSW42600056] files required	chapter 5.1
[BSW42600057] configuration and implementation	chapter 5.1

7 Functional specification



AUTOSAR Layered View [2].



AUTOSAR Layered View With CSM

7.1 Basic architecture guidelines

The starting point for the description of the design of the CSM module is the AUTOSAR Layered Software Architecture (see Figure AUTOSAR Layered View). The description of the CSM module architecture on the basis of the AUTOSAR layered software architecture shall help to understand the specification of interfaces and functionalities of the CSM module in the following sections.

The architecture of AUTOSAR consists of several layers which can be seen in Figure AUTOSAR Layered View. The Service Layer is the highest layer of the Basic Software. Its task is to provide basic services for application and basic software modules, i.e. it offers the most relevant functionalities for application software and basic software modules.

CSM is a service that provides cryptography functionality, based on a software library (above: “basic crypto routines”) or on a hardware module (above: “Crypto HW”). Also, mixed setups are possible, for example if the hardware module cannot supply the necessary functionality on its own. In the following, we refer to all instantiations of underlying functionality, be it hardware or software, as “crypto library”.

Note that hardware modules may include secure key storage capability. This means that secrets are protected by hardware means and can thus not be read or altered by malicious software. To be able to support this feature, the defined key types have been altered in order to allow the specification of key handles instead of raw key data. Furthermore, parameters that are used for key output are INOUT to allow the specification of a target handle and such by the caller.

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.

CSM0015

Due to memory restrictions the CSM module 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 CSM module have to be generated based on a configuration that describes which cryptographic methods are necessary for the applications.

7.2 General behavior

CSM0016

The CSM module shall only support processing of a single instance of each service at a time.

[1] CPL is defined by the Crypto Abstraction Library (see [15])

CSM0022

The CSM module shall allow parallel access to different services.

CSM0017

If a service of the CSM module is requested and this service is not idle (processing currently performed), the CSM module shall reject the service request by letting the interface function return the value CSM_E_BUSY.

CSM0019

If an asynchronous interface is configured, the CSM module shall provide a main function Csm_MainFunction() which is called cyclically to control the processing of the services via a state machine.

CSM0020

If interruption of job processing is configured, the CSM module shall provide a interruption function Csm_Interruption() which can be called to interrupt the processing of the services.

CSM0021

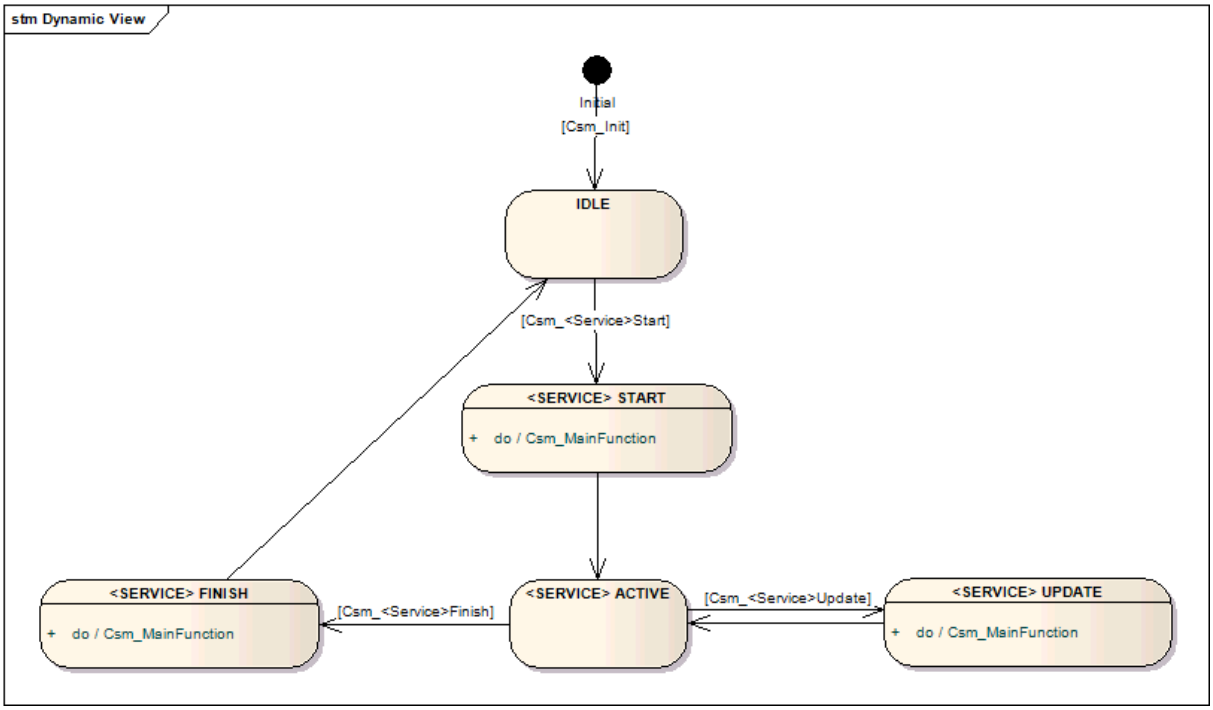
The function Csm_Init() shall initialize all variables used by the CSM module to an initial state.

7.2.1 Normal operation

CSM0023

The implementation of those CSM 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 CSM0024 shows the general design of such a CSM service.

CSM0024



7.2.2 Functional requirements

7.2.2.1 Configuration

CSM0025

Each service configuration shall be realized as a constant structure of type Csm_<Service>ConfigType .

CSM0026

Each service configuration shall have a name which can be configured.

CSM0028

It shall be possible to create arbitrary many service configurations for each cryptographic service.

CSM0029

When creating a service configuration, it shall be possible to configure all available and allowed schemes and underlying cryptographic primitives.

CSM0030

It shall be checked during configuration that only valid service configurations are chosen.

CSM0031

It shall be possible to configure synchronous or asynchronous job processing.

CSM0032

If the asynchronous interface is chosen, each service configuration shall contain a callback function.

CSM0033

If the asynchronous interface is chosen, it shall be possible to configure interruption of job processing.

7.2.2.2 Synchronous job processing

CSM0035

When the synchronous interface is used, the interface functions shall immediately compute the result.

7.2.2.3 Asynchronous job processing

CSM0036

If the asynchronous interface is used, the interface functions shall only hand over the necessary information to the service. The actual computation shall be done by the main function.

CSM0037

For each asynchronous request a notification of the caller after completion of the job shall be a configurable option. A callback interface has to be provided by the CSM module.

CSM0038

The Csm_MainFunction shall perform the processing of the services of the CSM module.

CSM0039

The users of the CSM shall be notified when a requested cryptographic service has been processed by calling the callback function from the service configuration.

7.2.2.4 Interruption of job processing

CSM0648

Interruption of job processing shall only be possible if the asynchronous interface is chosen.

CSM0649

If interruption of job processing is configured, the CSM shall provide a configuration option to configure the maximum time a service is allowed to run.

CSM0650

The minimum value of this time limit is implementation dependent.

CSM0651

If interruption of job processing is configured, the CSM and the underlying CRY module shall check during execution of a service, whether the computation shall be interrupted.

CSM0652

The computation of a service shall be interrupted, if the computation time has reached the configured time limit.

CSM0653

The computation of a service shall be interrupted, if the user calls the function Csm_Interruption(). In this case the time until interruption shall not exceed the minimum time limit.

CSM0654

If interruption of a service computation is indicated, the service shall return to the main function.

CSM0655

If the computation of a service was interrupted, it shall be resumed with the next call of the main function.

CSM0656

If interruption is configured, it affects all running services. It is only possible to configure interruption globally for all services.

7.2.3 Design notes

7.2.3.1 CSM module startup

The function Csm_Init shall be invoked by the ECU state manager exclusively. The Csm_Init request shall not be responsible to trigger the initialization of the underlying crypto library. This shall also be handled by the ECU state manager.

Software components which are using the CSM module shall be responsible for checking global error and status information resulting from the CSM module startup.

7.2.3.2 Synchronisation between application and CSM module

CSM0734

CSM services, which do not expect arbitrary amounts of user data, only have to provide an API Csm_<Service>() (e.g. Csm_RandomGenerate). These services shall be handled as simple function calls.

CSM services, which expect arbitrary amounts of user data, shall provide the APIs Csm_<Service>Start(), Csm_<Service>Update() and Csm_<Service>Finish(). To minimize locking and unlocking overhead or the use of other synchronization methods for CSM services which expect arbitrary amounts of user data, the communication between applications and these CSM services shall follow a strict sequence of steps which is described below. This ensures a reliable communication between applications and the CSM module.

All applications have to keep with the following rules:

7.2.3.2.1 Initialization

CSM0046

The application calls the Csm_<Service>Start request, passing a valid service configuration to the start function. The start function shall check the validity of the configuration it receives.

CSM0537

If an instance of this service is being processed when Csm_<Service>Start is called, the function shall return CSM_E_BUSY.

CSM0047

If the synchronous interface is used, and no instance of this service is being processed when Csm_<Service>Start is called, the function shall configure the CSM immediately, set the status of the current service to active, and return the status of the service.

CSM0048

If the asynchronous interface is used, and no instance of this service is being processed when Csm_<Service>Start is called, the function shall return with CSM_E_OK. The main function shall process the actual initialization and set the status of the current service to active. After completing the initialization the main function shall call the callback function as given in the service configuration.

7.2.3.2.2 Update

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

CSM0050

The application calls the Csm_<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.

CSM0657

If the service has not been initialized before, the update function shall return with CSM_E_NOT_OK.

CSM0051

The CSM shall assume that the data provided to Csm_<Service>Update will not change until it returns in case of synchronous processing, or until the callback function is called in case of asynchronous processing.

CSM0052

If the synchronous interface is used, and 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, and return the status of the update.

CSM0053

If the asynchronous interface is used, and the service has been initialized before, the update function shall return with CSM_E_OK. The main function shall process the actual data update and set the status of the service again to active. After completing the update the main function shall call the callback function as given in the service configuration.

CSM0054

The CSM shall allow the application to call the update function arbitrarily often.

7.2.3.2.3 Finish

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

CSM0056

The application calls the Csm_<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.

CSM0658

If,the service has not been initialized before, the finish function shall return with CSM_E_NOT_OK.

CSM0057

The CSM shall assume that the data provided to Csm_<Service>Finish will not change until it returns in case of synchronous processing, or until the callback function is called in case of asynchronous processing.

CSM0058

If the synchronous interface is used, and the service has been initialized before, the finish function shall immediately process the given data, finish the computation of the current cryptographic service, store the result of the service in the result buffer, set the status of the service to idle, and return the status of the finishing.

CSM0059

If the asynchronous interface is used, and the service has been initialized before, the finish function shall return with CSM_E_OK. The main function shall process the actual result computation and storage, and set the status of the service to idle. After completing this computation the main function shall call the callback function as given in the service configuration.

7.3 Version check

CSM0060
The CSM module shall perform Inter Module Checks to avoid integration of incompatible files.
The imported included files shall be checked by preprocessing directives.

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 CSM module.

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

7.4 Error classification

CSM0538

Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem.h.

CSM0664

Development error values are of type uint8.

Type of error	*Relevance*	*Related error code*	*Value [hex]*
API request called with invalid parameter (Nullpointer)	Development	`CSM_E_PARAM_PTR_INVALID`	0x01

Requested service is not initialized	Development	CSM_E_SERVICE_NOT_STARTED	0x02
API request called with invalid parameter (invalid method for selected service)	Development	CSM_E_PARAM_METHOD_INVALID	0x03
API request called with invalid parameter (invalid key type for selected service)	Development	CSM_E_PARAM_KEY_TYPE_INVALID	0x04
API request called before initialisation of CSM module	Development	CSM_E_UNINIT	0x05
Provided buffer for storing the result of a computation is too small.	Development	CSM_E_BUFFER_TOO_SMALL	0x06
Initialization of CSM module failed	Production	CSM_E_INIT_FAILED	Assigned by DEM
API request violated the security policy, e.g. by trying to overwrite a key that is read-only	Production	CSM_E_POLICY_VIOLATION	Assigned by DEM

7.5 Error detection

CSM0062

The detection of development error shall be configurable (ON/OFF) at compile time. The switch CSM_DEV_ERROR_DETECT shall activate or deactivate the detection of all development errors.

CSM0063

If the CSM_DEV_ERROR_DETECT switch is enabled, the API parameters shall be checked in the order in which they are passed.

CSM0065

The detection of production code errors cannot be switched off.

CSM0488

If the development error detection is enabled and an error is detected, the desired service shall return with CSM_E_NOT_OK.

CSM0489

The following table specifies which DET error values shall be reported for each API call:

CSM0539

API call	*Error condition*	*DET related error value*
Csm_<Service>Start	CSM is not initialized	CSM_E_UNINIT
Csm_MainFunction		
Csm_Interruption		
All APIs that have a pointer as parameter	Pointer is Nullpointer	CSM_E_PARAM_PTR_INVALID In case of this error, the API service shall return immediately without any further action, beside reporting this development error.
Csm_<Service>Update	Service is not initialized	CSM_E_SERVICE_NOT_STARTED
Csm_<Service>Finish	Service is not initialized	CSM_E_SERVICE_NOT_STARTED
Csm_<Service>Start	Invalid cryptographic method for selected service	CSM_E_PARAM_METHOD_INVALID
Csm_MacGenerateStart	Invalid key type for selected service	CSM_E_PARAM_KEY_TYPE_INVALID
Csm_MacVerifyStart		
Csm_SymBlockEncryptStart		
Csm_SymBlockDecryptStart		
Csm_SymEncryptStart		
Csm_SymDecryptStart		
Csm_AsymEncryptStart		
Csm_AsymDecryptStart		
Csm_SigGenerateStart		
Csm_SigVerifyStart		
Csm_KeyDeriveStart		
Csm_KeyDeriveSymKey		
Csm_KeyExchangeCalcPubVal		
Csm_KeyExchangeCalcSecretStart		
Csm_SymKeyExtractStart		
Csm_SymKeyWrapSymStart		
Csm_SymKeyWrapAsymStart		
Csm_AsymPrivateKeyExtractStart		
Csm_AsymPrivateKeyWrapSymStart		
Csm_AsymPrivateKeyWrapAsymStart		
Csm_AsymPublicKeyExtractStart		

7.6 Error notification

CSM0066

Production errors shall be reported to the DEM.

CSM0067

Detected development errors shall be reported to the Det_ReportError service of the DET if the switch CSM_DEV_ERROR_DETECT is enabled.

CSM0774

If CSM detects a security policy violation, the production error CSM_E_POLICY_VIOLATION shall be reported to DEM and the service return code shall be CSM_E_NOT_OK.

7.7 Debugging concept

CSM0696

Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable.

CSM0697

All type definitions of variables which shall be debugged, shall be accessible by the header file Csm.h.

CSM0698

The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-Language "sizeof".

CSM0699

Variables available for debugging shall be described in the respective Basic Software Module Description.

CSM0542

The states of the CSM state machine shall be available for debugging.

8 API specification

8.1 Imported types

CSM0068

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 Csm_ReturnType

CSM0069

Name:	`Csm_ReturnType`	
Type:	`Enumeration`	
Range:	`CSM_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"
	`CSM_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".
	`CSM_E_BUSY`	the service request failed because the service is still busy. This return code shall be given as value "2".
	`CSM_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".
	`CSM_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 CSM module	

8.2.1.2 Csm_CallbackType

CSM0073

Name:	`Csm_CallbackType`
Type:	`Std_FunctionPointer`
Description:	Function pointer for the callback function which will be invoked after service completion. Signature: Std_ReturnType (*Csm_CallbackType)(Csm_ReturnType)

8.2.1.3 Csm_ConfigIdType

CSM0691

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

8.2.1.4 Csm_<Service>ConfigType

CSM0074

Name:	Csm_<Service>ConfigType		
Type:	Structure		
Element:	Csm_ConfigIdType	ConfigId	The numeric identifier of a configuration.
	Csm_CallbackType	CallbackFct	A pointer to the callback function which shall be called when the configured service has finished. This Element is only available if "CsmUseSyncJobProcessing" is "OFF". (CSM0557_Conf)
	Csm_ReturnType	*PrimitiveStartFct(<primitive parameter list>)	This element shall only exist if the service contains the function Csm_<Service>Start. It is a pointer to the function Cry_<Primitive>Start of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cry_<Primitive>Start.
	Csm_ReturnType	*PrimitiveUpdateFct (<primitive parameter list>)	This element shall only exist if the service contains the function Csm_<Service>Update. It is a pointer to the function Cry_<Primitive>Update of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cry_<Primitive>Update.
	Csm_ReturnType	*PrimitiveFinishFct(<primitive parameter list>)	This element shall only exist if the service contains the function Csm_<Service>Finish. It is a pointer to the function Cry_<Primitive>Finish of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cry_<Primitive>Finish.
	Csm_ReturnType	*PrimitiveFct (<primitive parameter list>)	This element shall only exist if the service contains the

			function Csm_<Service>. It is a pointer to the function Cry_<Primitive> of the configured cryptographic primitive. For the "primitive parameter list" see the description of Cry_<Primitive>.
	void	*PrimitiveMainFct (void)	a pointer to the function Cry_<Primitive>MainFunction of the configured cryptographic 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.		

8.2.1.5 Csm_AlignType

CSM0728

Name:	Csm_AlignType
Type:	<maxAlignScalarType>
Description:	A scalar type which has maximum alignment restrictions on the given platform. This value is configured by "CsmMaxAlignScalarType". <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.

8.2.1.6 Csm_VerifyResultType

CSM0075

Name:	Csm_VerifyResultType	
Type:	Enumeration	
Range:	CSM_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"
	CSM_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.	

8.2.1.7 Csm_AsymPublicKeyType

CSM0079

Name:	Csm_AsymPublicKeyType

Type:	Structure		
Element:	uint32	length	This element contains the length of the key stored in element 'data'
	Csm_AlignType [CSM_ASYM_PUB_KEY_MAX_SIZE]	data	This element contains the key data or a key handle.
Description:	Structure for the public asymmetrical key. CSM_ASYM_PUB_KEY_MAX_SIZE shall be chosen such that "CSM_ASYM_PUB_KEY_MAX_SIZE * sizeof(Csm_AlignType)" is greater or equal to the maximum of the configured values CsmAsymEncryptMaxKeySize, CsmSignatureVerifyMaxKeySize, CsmAsymPublicKeyExtractMaxKeySize, CsmSymKeyWrapAsymMaxPubKeySize and CsmAsymPrivateKeyWrapAsymPubKeySize.		

8.2.1.8 Csm_AsymPrivateKeyType

CSM0080

Name:	Csm_AsymPrivateKeyType		
Type:	Structure		
Element:	uint32	length	This element contains the length of the key stored in element 'data'
	Csm_AlignType [CSM_ASYM_PRIV_KEY_MAX_SIZE]	data	This element contains the key data or a key handle.
Description:	Structure for the private asymmetrical key. CSM_ASYM_PRIV_KEY_MAX_SIZE shall be chosen such that "CSM_ASYM_PRIV_KEY_MAX_SIZE * sizeof(Csm_AlignType)" is greater or equal to the maximum of the configured values CsmAsymDecryptMaxKeySize, CsmSignatureGenerateMaxKeySize, CsmAsymPrivateKeyExtractMaxKeySize, CsmAsymPrivateKeyWrapSymMaxPrivKeySize and CsmAsymPrivateKeyWrapAsymMaxPrivKeySize.		

8.2.1.9 Csm_SymKeyType

CSM0082

Name:	Csm_SymKeyType		
Type:	Structure		
Element:	uint32	length	This element contains the length of the key stored in element 'data'
	Csm_AlignType [CSM_SYM_KEY_MAX_SIZE]	data	This element contains the key data or a key handle.
Description:	Structure for the symmetrical key. CSM_SYM_KEY_MAX_SIZE shall be chosen such that "CSM_SYM_KEY_MAX_SIZE * sizeof(Csm_AlignType)" is greater or equal to the maximum of the configured values CsmSymBlockEncryptMaxKeySize, CsmSymBlockDecryptMaxKeySize, CsmSymEncryptMaxKeySize, CsmSymDecryptMaxKeySize, CsmKeyDeriveMaxKeySize, CsmSymKeyExtractMaxKeySize, CsmMacGenerateMaxKeySize and CsmMacVerifyMaxKeySize, CsmSymKeyWrapSymMaxSymKeySize, CsmSymKeyWrapAsymMaxSymKeySize, CsmAsymPrivateKeyWrapSymMaxSymKeySize, CsmKeyExchangeCalcSymKeyMaxSymKeySize and CsmKeyDeriveSymKeyMaxSymKeySize.		

8.2.1.10 Csm_KeyExchangeBaseType

CSM0086

Name:	`Csm_KeyExchangeBaseType`		
Type:	`Structure`		
Element:	`uint32`	`length`	This element contains the length of the key stored in element 'data'
	`Csm_AlignType [CSM_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. CSM_KEY_EX_BASE_MAX_SIZE shall be chosen such that "CSM_KEY_EX_BASE_MAX_SIZE * sizeof(Csm_AlignType)" is greater or equal to the maximum of the configured values CsmKeyExchangeCalcPubValMaxBaseTypeSize, CsmKeyExchangeCalcSecretMaxBaseTypeSize and CsmKeyExchangeCalcSymKeyMaxBaseTypeSize.		

8.2.1.11 Csm_KeyExchangePrivateType

CSM0087

Name:	`Csm_KeyExchangePrivateType`		
Type:	`Structure`		
Element:	`uint32`	`length`	This element contains the length of the key stored in element 'data'
	`Csm_AlignType [CSM_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. CSM_KEY_EX_PRIV_MAX_SIZE shall be chosen such that "CSM_KEY_EX_PRIV_MAX_SIZE * sizeof(Csm_AlignType)" is greater or equal to the maximum of the configured values CsmKeyExchangeCalcPubValMaxPrivateTypeSize, CsmKeyExchangeCalcSecretMaxPrivateTypeSize CsmKeyExchangeCalcSymKeyMaxPrivateTypeSize.		

8.3 API functions

CSM0478

All functions need not to be reentrant. For behaviour in case of a reentrant call see [CSM0017].

8.3.1 General interfaces

8.3.1.1 Csm_Init

CSM0646

Service name:	Csm_Init

Syntax:	`void Csm_Init(`
	`  void`
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This function shall be used to initialize the CSM module.

CSM0659

If the initialization of the CSM module fails, the CSM shall report `CSM_E_INIT_FAILED` to the DEM.

8.3.1.2 Csm_GetVersionInfo

CSM0705:

Service name:	Csm_GetVersionInfo	

Syntax:	`void Csm_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:	None	
Description:	Returns the version information of this module.	

CSM0706:
The function Csm_GetVersionInfo shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

CSM0707:
The function Csm_GetVersionInfo shall be pre compile time configurable On/Off by the configuration parameter: CSM_VERSION_INFO_API

8.3.1.3 Csm_Interruption

CSM0647

Service name:	Csm_Interruption

Syntax:	`void Csm_Interruption(`
	`  void`
	`)`

Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	If interruption of the Csm is turned on with the configuration macro "CsmUseInterruption", this function shall be used to interrupt a currently running Csm main function. After this function is called, the main function will interrupt itself after a time which does not exceed the minimal allowed value for the configuration option "CsmMaximumBlockingTime".

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 Csm_HashStart

CSM0089

Service name:	Csm_HashStart	

Syntax:	`Csm_ReturnType Csm_HashStart(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration that has to be used during the hash value computation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the hash service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active"	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_HashStart.

8.3.2.2 Csm_HashUpdate

CSM0094

Service name:	Csm_HashUpdate	

Syntax:	`Csm_ReturnType Csm_HashUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	dataPtr	Holds a pointer to the data to be hashed
	dataLength	Contains the number of bytes to be hashed.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the hash service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The hash computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_HashUpdate.

8.3.2.3 Csm_HashFinish

CSM0101

Service name:	Csm_HashFinish	

Syntax:	`Csm_ReturnType Csm_HashFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* resultPtr,`
	`  uint32* resultLengthPtr,`
	`  boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.

	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):	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. When the request has finished, the actual length of the returned 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result, and truncation was not allowed.
Description:	This interface shall be used to finish the hash service of the CSM module. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The hash computation is done by the underlying primitive.	

CSM0661

The CSM 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 `CSM_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_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 Csm_MacGenerateStart

CSM0108

Service name:	Csm_MacGenerateStart	

Syntax:	`Csm_ReturnType Csm_MacGenerateStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the MAC generate service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_MacGenerateStart.

8.3.3.2 Csm_MacGenerateUpdate

CSM0114

Service name:	Csm_MacGenerateUpdate	

Syntax:	`Csm_ReturnType Csm_MacGenerateUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`	

Service ID[hex]:	0x07	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	None	

Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the MAC generate service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_MacGenerateUpdate.

8.3.3.3 Csm_MacGenerateFinish

CSM0121

Service name:	Csm_MacGenerateFinish	

Syntax:	`Csm_ReturnType Csm_MacGenerateFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* resultPtr,`
	`  uint32* resultLengthPtr,`
	`  boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	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. When the request has finished, the actual length of the returned 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result, and truncation was not allowed.
Description:	This interface shall be used to finish the MAC generation service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK".	

	Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.

CSM0662

The CSM 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 `CSM_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_MacGenerateFinish.

8.3.3.4 Csm_MacVerifyStart

CSM0128

Service name:	Csm_MacVerifyStart	

Syntax:	`Csm_ReturnType Csm_MacVerifyStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the MAC verify service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_MacVerifyStart.

8.3.3.5 Csm_MacVerifyUpdate

CSM0134

Service name:	Csm_MacVerifyUpdate	

Syntax:	`Csm_ReturnType Csm_MacVerifyUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`	

Service ID[hex]:	0x0a	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the MAC verification service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_MacVerifyUpdate.

8.3.3.6 Csm_MacVerifyFinish

CSM0141

Service name:	Csm_MacVerifyFinish	

Syntax:	`Csm_ReturnType Csm_MacVerifyFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* MacPtr,`
	`  uint32 MacLength,`
	`  Csm_VerifyResultType* resultPtr`
	`)`	

Service ID[hex]:	0x0b	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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. Note: the computed MAC will be internally truncated to this length.
Parameters (inout):	None	
Parameters (out):	resultPtr	holds a pointer to the memory location which will hold the result of the MAC verification.
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the MAC verification service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_MacVerifyFinish.

8.3.4 Random interface

The random interface provides generation of random numbers. A random number can be generated either by a physical device (true random number generator), or by computational algorithms (pseudo random number generator). The randomness of pseudo random number generators can be increased by an appropriate selection of the seed.

8.3.4.1 Csm_RandomSeedStart

CSM0149

Service name:	Csm_RandomSeedStart	

Syntax:	`Csm_ReturnType Csm_RandomSeedStart(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x0c	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration which has to be used during the seeding of the random number generator.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the random seed service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY".	

	Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_RandomSeedStart.

8.3.4.2 Csm_RandomSeedUpdate

CSM0156

Service name:	Csm_RandomSeedUpdate	

Syntax:	`Csm_ReturnType Csm_RandomSeedUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* seedPtr,`
	`  uint32 seedLength`
	`)`	

Service ID[hex]:	0x0d	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	seedPtr	holds a pointer to the seed for the random number generator.
	seedLength	contains the length of the seed in bytes.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed a seed to the random seed service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_RandomSeedUpdate.

8.3.4.3 Csm_RandomSeedFinish

CSM0163

Service name:	Csm_RandomSeedFinish

Syntax:	`Csm_ReturnType Csm_RandomSeedFinish(`
	`  Csm_ConfigIdType cfgId`
	`)`

Service ID[hex]:	0x0e
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)

Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the random seed service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_RandomSeedFinish.

8.3.4.4 Csm_RandomGenerate

CSM0543

Service name:	Csm_RandomGenerate	

Syntax:	`Csm_ReturnType Csm_RandomGenerate(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* resultPtr,`
	`  uint32 resultLength`
	`)`	

Service ID[hex]:	0x0f	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration which has to be used during random number generation.
	resultLength	holds the amount of random bytes which should be generated.
Parameters (inout):	None	
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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_ENTROPY_EXHAUSTION: request failed, entropy of randum number generator is exhausted.
Description:	This interface shall be used to start the random number generation service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive> of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive> returned successfully, the service state has to be set to "active". The random number generation is done by the underlying primitive.	

The generation of a random number is based on the seed, which was previously set with the interfaces Csm_RandomSeedStart, Csm_RandomSeedUpdate, and Csm_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 Csm_RandomGenerate can be called arbitrarily often to generate multiple random numbers.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_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 Csm_SymBlockEncryptStart

CSM0168

Service name:	Csm_SymBlockEncryptStart	

Syntax:	`Csm_ReturnType Csm_SymBlockEncryptStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x10	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the symmetrical block encrypt service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymBlockEncryptStart.

8.3.5.2 Csm_SymBlockEncryptUpdate

CSM0173

Service name:	Csm_SymBlockEncryptUpdate	

Syntax:	`Csm_ReturnType Csm_SymBlockEncryptUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* plainTextPtr,`
	`  uint32 plainTextLength,`
	`  uint8* cipherTextPtr,`
	`  uint32* cipherTextLengthPtr`
	`)`	

Service ID[hex]:	0x11	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	plainTextPtr	holds a pointer to the plain text that shall be encrypted.
	plainTextLength	contains the length of the plain text in bytes
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to feed the symmetrical block encryption service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The encryption process is done by the underlying primitive.	

CSM0663

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymBlockEncryptUpdate.

8.3.5.3 Csm_SymBlockEncryptFinish

CSM0180

Service name:	Csm_SymBlockEncryptFinish	

Syntax:	`Csm_ReturnType Csm_SymBlockEncryptFinish(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x12	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the symmetrical block encryption service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The encryption process is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymBlockEncryptFinish.

8.3.5.4 Csm_SymBlockDecryptStart

CSM0187

Service name:	Csm_SymBlockDecryptStart	

Syntax:	`Csm_ReturnType Csm_SymBlockDecryptStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x13	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the constant CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the symmetrical block decrypt service of the CSM module.	

	If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymBlockDecryptStart.

8.3.5.5 Csm_SymBlockDecryptUpdate

CSM0192

Service name:	Csm_SymBlockDecryptUpdate	

Syntax:	`Csm_ReturnType Csm_SymBlockDecryptUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* cipherTextPtr,`
	`  uint32 cipherTextLength,`
	`  uint8* plainTextPtr,`
	`  uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x14	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to feed the symmetrical block decryption service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The decryption process is done by the underlying primitive.	

CSM0700

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymBlockDecryptUpdate.

8.3.5.6 Csm_SymBlockDecryptFinish

CSM0199

Service name:	Csm_SymBlockDecryptFinish	
Syntax:	`Csm_ReturnType Csm_SymBlockDecryptFinish( Csm_ConfigIdType cfgId )`	
Service ID[hex]:	0x15	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the symmetrical block decryption service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The decryption process is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_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 Csm_SymEncryptStart

CSM0206

Service name:	Csm_SymEncryptStart
Syntax:	`Csm_ReturnType Csm_SymEncryptStart( Csm_ConfigIdType cfgId,`

	`const Csm_SymKeyType* keyPtr,`
	`const uint8* InitVectorPtr,`
	`uint32 InitVectorLength`
	)	

Service ID[hex]:	0x16	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the symmetrical encrypt service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymEncryptStart.

8.3.6.2 Csm_SymEncryptUpdate

CSM0212

Service name:	Csm_SymEncryptUpdate	

Syntax:	`Csm_ReturnType Csm_SymEncryptUpdate(`
	`Csm_ConfigIdType cfgId,`
	`const uint8* plainTextPtr,`
	`uint32 plainTextLength,`
	`uint8* cipherTextPtr,`
	`uint32* cipherTextLengthPtr`
	)	

Service ID[hex]:	0x17	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	plainTextPtr	holds a pointer to the plain text that shall be encrypted.
	plainTextLength	contains the length of the plain text in bytes
Parameters	cipherTextLengthPtr	holds a pointer to a memory location in which the length

(inout):		information is stored. On calling this function this parameter shall contain the size of the buffer provided by cipherTextPtr. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to feed the symmetrical encryption service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The encryption process is done by the underlying primitive.	

CSM0665

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymEncryptUpdate.

8.3.6.3 Csm_SymEncryptFinish

CSM0221

Service name:	Csm_SymEncryptFinish	

Syntax:	`Csm_ReturnType Csm_SymEncryptFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* cipherTextPtr,`
	`  uint32* cipherTextLengthPtr`
	`)`	

Service ID[hex]:	0x18	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to finish the symmetrical encryption service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The encryption process is done by the underlying primitive.	

CSM0666

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymEncryptFinish.

8.3.6.4 Csm_SymDecryptStart

CSM0228

Service name:	Csm_SymDecryptStart	

Syntax:	`Csm_ReturnType Csm_SymDecryptStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* keyPtr,`
	`  const uint8* InitVectorPtr,`
	`  uint32 InitVectorLength`
	`)`	

Service ID[hex]:	0x19	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the symmetrical decryption service of the CSM module.	

	If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymDecryptStart.

8.3.6.5 Csm_SymDecryptUpdate

CSM0234

Service name:	Csm_SymDecryptUpdate	

Syntax:	`Csm_ReturnType Csm_SymDecryptUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* cipherTextPtr,`
	`  uint32 cipherTextLength,`
	`  uint8* plainTextPtr,`
	`  uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x1a	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	cipherTextPtr	holds a pointer to the cipher text that shall be decrypted.
	cipherTextLength	contains the length of the cipher text in bytes.
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to feed the symmetrical decryption service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The decryption process is done by the underlying primitive.	

CSM0667

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymDecryptUpdate.

8.3.6.6 Csm_SymDecryptFinish

CSM0243

Service name:	Csm_SymDecryptFinish	

Syntax:	`Csm_ReturnType Csm_SymDecryptFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* plainTextPtr,`
	`  uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x1b	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to finish the symmetrical decryption service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The decryption process is done by the underlying primitive.	

CSM0668

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_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 Csm_AsymEncryptStart

CSM0250

Service name:	Csm_AsymEncryptStart	

Syntax:	`Csm_ReturnType Csm_AsymEncryptStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_AsymPublicKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x1c	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration which has to be used during the asymmetrical encryption computation
	keyPtr	holds a pointer to the key necessary for the asymmetrical computation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the asymmetrical encryption service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymEncryptStart.

8.3.7.2 Csm_AsymEncryptUpdate

CSM0256

Service name:	Csm_AsymEncryptUpdate

Syntax:	`Csm_ReturnType Csm_AsymEncryptUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* plainTextPtr,`
	`  uint32 plainTextLength,`
	`  uint8* cipherTextPtr,`
	`  uint32* cipherTextLengthPtr`

	)	
Service ID[hex]:	0x1d	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	plainTextPtr	holds a pointer to the plain text that shall be encrypted.
	plainTextLength	contains the length of the plain text in bytes
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to feed the asymmetrical encryption service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The encryption process is done by the underlying primitive.	

CSM0669

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymEncryptUpdate.

8.3.7.3 Csm_AsymEncryptFinish

CSM0265

Service name:	Csm_AsymEncryptFinish	

Syntax:	`Csm_ReturnType Csm_AsymEncryptFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* cipherTextPtr,`
	`  uint32* cipherTextLengthPtr`
	)	

Service ID[hex]:	0x1e	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters	cipherTextLengthPtr	holds a pointer to a memory location in which the length

(inout):		information is stored. On calling this function this parameter shall contain the size of the buffer provided by cipherTextPtr. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to finish the asymmetrical encryption service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The encryption process is done by the underlying primitive.	

CSM0670

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymEncryptFinish.

8.3.7.4 Csm_AsymDecryptStart

CSM0272

Service name:	Csm_AsymDecryptStart	

Syntax:	`Csm_ReturnType Csm_AsymDecryptStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_AsymPrivateKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x1f	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration which has to be used during the asymmetrical decryption computation
	keyPtr	holds a pointer to the key necessary for the asymmetrical computation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the asymmetrical decryption service of the CSM module.	

	If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymDecryptStart.

8.3.7.5 Csm_AsymDecryptUpdate

CSM0278

Service name:	Csm_AsymDecryptUpdate	

Syntax:	`Csm_ReturnType Csm_AsymDecryptUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* cipherTextPtr,`
	`  uint32 cipherTextLength,`
	`  uint8* plainTextPtr,`
	`  uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x20	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	cipherTextPtr	holds a pointer to the encrypted data
	cipherTextLength	contains the length of the encrypted data in bytes.
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to feed the asymmetrical decryption service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The decryption process is done by the underlying primitive.	

CSM0671

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymDecryptUpdate.

8.3.7.6 Csm_AsymDecryptFinish

CSM0287

Service name:	Csm_AsymDecryptFinish	

Syntax:	`Csm_ReturnType Csm_AsymDecryptFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* plainTextPtr,`
	`  uint32* plainTextLengthPtr`
	`)`	

Service ID[hex]:	0x21	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to finish the asymmetrical decryption service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The decryption process is done by the underlying primitive.	

CSM0672

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_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 Csm_SignatureGenerateStart

CSM0294

Service name:	Csm_SignatureGenerateStart	

Syntax:	`Csm_ReturnType Csm_SignatureGenerateStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_AsymPrivateKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x22	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the signature generate service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SignatureGenerateStart.

8.3.8.2 Csm_SignatureGenerateUpdate

CSM0300

Service name:	Csm_SignatureGenerateUpdate

Syntax:	`Csm_ReturnType Csm_SignatureGenerateUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`

	)	
Service ID[hex]:	0x23	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	dataPtr	holds a pointer to the data that shall be signed
	dataLength	contains the length of the data to be signed
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the signature generation service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SignatureGenerateUpdate.

8.3.8.3 Csm_SignatureGenerateFinish

CSM0307

Service name:	Csm_SignatureGenerateFinish	

Syntax:	`Csm_ReturnType Csm_SignatureGenerateFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* resultPtr,`
	`  uint32* resultLengthPtr`
	)	

Service ID[hex]:	0x24	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to finish the signature generation service.	

	If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.

CSM0673

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SignatureGenerateFinish.

8.3.8.4 Csm_SignatureVerifyStart

CSM0314

Service name:	Csm_SignatureVerifyStart	

Syntax:	`Csm_ReturnType Csm_SignatureVerifyStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_AsymPublicKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x25	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the signature verify service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SignatureVerifyStart.

8.3.8.5 Csm_SignatureVerifyUpdate

CSM0320

Service name:	Csm_SignatureVerifyUpdate	

Syntax:	`Csm_ReturnType Csm_SignatureVerifyUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`	

Service ID[hex]:	0x26	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	dataPtr	holds a pointer to the signature which shall be verified
	dataLength	contains the length of the signature to verify in bytes
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the signature verification service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SignatureVerifyUpdate.

8.3.8.6 Csm_SignatureVerifyFinish

CSM0327

Service name:	Csm_SignatureVerifyFinish	

Syntax:	`Csm_ReturnType Csm_SignatureVerifyFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* signaturePtr,`
	`  uint32 signatureLength,`
	`  Csm_VerifyResultType* resultPtr`
	`)`	

Service ID[hex]:	0x27	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	None	
Parameters (out):	resultPtr	holds a pointer to the memory location which will hold the result of the signature verification.
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the signature verification service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SignatureVerifyFinish.

8.3.9 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.

8.3.9.1 Csm_ChecksumStart

CSM0335

Service name:	Csm_ChecksumStart	

Syntax:	`Csm_ReturnType Csm_ChecksumStart(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x28	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration which has to be used during the checksum computation
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the checksum service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_ChecksumStart.

8.3.9.2 Csm_ChecksumUpdate

CSM0341

Service name:	Csm_ChecksumUpdate	

Syntax:	`Csm_ReturnType Csm_ChecksumUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`	

Service ID[hex]:	0x29	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the checksum service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The checksum computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_ChecksumUpdate.

8.3.9.3 Csm_ChecksumFinish

CSM0348

Service name:	Csm_ChecksumFinish	

Syntax:	`Csm_ReturnType Csm_ChecksumFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* resultPtr,`
	`  uint32* resultLengthPtr,`
	`  boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x2a	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has

		to be used during the operation.
	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):	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. When the request has finished, 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, and truncation is allowed, the result shall be truncated
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result, and truncation was not allowed.
Description:	This interface shall be used to finish the checksum service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The checksum computation is done by the underlying primitive.	

CSM0674

The CSM 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 `CSM_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_ChecksumFinish.

8.3.10 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.
Specification of input keys that are protected by hardware means can be achieved by using the Csm_KeyDeriveSymKey interface.

8.3.10.1 Csm_KeyDeriveStart

CSM0355

Service name:	Csm_KeyDeriveStart

Syntax:	`Csm_ReturnType Csm_KeyDeriveStart(`
	`  Csm_ConfigIdType cfgId,`
	`  uint32 keyLength,`
	`  uint32 iterations`
	`)`	

Service ID[hex]:	0x2b	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the key derivation service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyDeriveStart.

8.3.10.2 Csm_KeyDeriveUpdate

CSM0362

Service name:	Csm_KeyDeriveUpdate	

Syntax:	`Csm_ReturnType Csm_KeyDeriveUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* passwordPtr,`
	`  uint32 passwordLength,`
	`  const uint8* saltPtr,`
	`  uint32 saltLength`
	`)`	

Service ID[hex]:	0x2c	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the key derivation service with the input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The key derivation computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyDeriveUpdate.

8.3.10.3 Csm_KeyDeriveFinish

CSM0371

Service name:	Csm_KeyDeriveFinish	

Syntax:	`Csm_ReturnType Csm_KeyDeriveFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  Csm_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x2d	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	keyPtr	holds a pointer to the memory location which will hold the result of the key derivation.
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the key derivation service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The key derivation computation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyDeriveFinish.

8.3.10.4 Csm_KeyDeriveSymKey

CSM0735

Service name:	Csm_KeyDeriveSymKey	

Syntax:	`Csm_ReturnType Csm_KeyDeriveSymKey(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* baseKeyPtr,`
	`  const uint8* customisationValPtr,`
	`  uint32 customisationValLength,`
	`  Csm_SymKeyType* derivedKeyPtr`
	`)`	

Service ID[hex]:	0x4c	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration which has to be used during the key derivation
	baseKeyPtr	holds a pointer to the key from which the new key shall be derived
	customisationValPtr	holds a pointer to the customisation value (if any)
	customisationValLength	holds the length of the customisation value in bytes
Parameters (inout):	derivedKeyPtr	holds a pointer to the memory location which will hold the result of the key derivation.
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the key derivation from key service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive> of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive> returned successfully, the service state has to be set to "active". The key derivation is done by the underlying primitive.	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyDeriveSymKey.

8.3.11 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.
Derivation of secret keys that shall be protected by hardware means can be done by using the Csm_KeyExchangeCalcSymKey interface.

8.3.11.1 Csm_KeyExchangeCalcPubVal

CSM0377

Service name:	Csm_KeyExchangeCalcPubVal
Syntax:	`Csm_ReturnType Csm_KeyExchangeCalcPubVal(` `  Csm_ConfigIdType cfgId,`

	const Csm_KeyExchangeBaseType* basePtr, const Csm_KeyExchangePrivateType* privateValuePtr, uint8* publicValuePtr, uint32* publicValueLengthPtr)	
Service ID[hex]:	0x2e	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result
Description:	This interface shall be used to start the public value calculation service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive> of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive> returned successfully, the service state has to be set to "active". The calculation of the public value is done by the underlying primitive.	

CSM0675

The CSM shall check if the provided buffer is large enough to hold the result of the computation. If the provided buffer is too small, CSM_E_SMALL_BUFFER shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyExchangeCalcPubVal.

8.3.11.2 Csm_KeyExchangeCalcSecretStart

CSM0396

Service name:	Csm_KeyExchangeCalcSecretStart
Syntax:	Csm_ReturnType Csm_KeyExchangeCalcSecretStart(

	Csm_ConfigIdType cfgId, const Csm_KeyExchangeBaseType* basePtr, const Csm_KeyExchangePrivateType* privateValuePtr)	
Service ID[hex]:	0x2f	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the key exchange service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyExchangeCalcSecretStart.

8.3.11.3 Csm_KeyExchangeCalcSecretUpdate

CSM0404

Service name:	Csm_KeyExchangeCalcSecretUpdate	
Syntax:	Csm_ReturnType Csm_KeyExchangeCalcSecretUpdate(Csm_ConfigIdType cfgId, const uint8* partnerPublicValuePtr, uint32 partnerPublicValueLength)	
Service ID[hex]:	0x30	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful

		CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface 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 is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_KeyExchangeCalcSecretUpdate.

8.3.11.4 Csm_KeyExchangeCalcSecretFinish

CSM0411

Service name:	Csm_KeyExchangeCalcSecretFinish	

Syntax:	`Csm_ReturnType Csm_KeyExchangeCalcSecretFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* sharedSecretPtr,`
	`  uint32* sharedSecretLengthPtr,`
	`  boolean TruncationIsAllowed`
	`)`	

Service ID[hex]:	0x31	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	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. When the request has finished, 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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the result provided buffer is too small to store the result, and truncation was not allowed.
Description:	This interface shall be used to finish the key exchange service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the	

	value returned by that function. The calculation of the shared secret is done by the underlying primitive.

CSM0676

The CSM 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 `CSM_E_OK` shall be returned. If the provided buffer is too small, and truncation is not allowed, `CSM_E_SMALL_BUFFER` shall be returned.

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyExchangeCalcSecretFinish.

8.3.11.5 Csm_KeyExchangeCalcSymKeyStart

CSM0736

Service name:	Csm_KeyExchangeCalcSymKeyStart	

Syntax:	`Csm_ReturnType Csm_KeyExchangeCalcSymKeyStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_KeyExchangeBaseType* basePtr,`
	`  const Csm_KeyExchangePrivateType* privateValuePtr`
	`)`	

Service ID[hex]:	0x3d	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the key exchange service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_KeyExchangeCalcSymKeyStart.

8.3.11.6 Csm_KeyExchangeCalcSymKeyUpdate

CSM0737

Service name:	Csm_KeyExchangeCalcSymKeyUpdate	

Syntax:	`Csm_ReturnType Csm_KeyExchangeCalcSymKeyUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* partnerPublicValuePtr,`
	`  uint32 partnerPublicValueLength`
	`)`	

Service ID[hex]:	0x3e	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	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):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface 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 is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_KeyExchangeCalcSymKeyUpdate.

8.3.11.7 Csm_KeyExchangeCalcSymKeyFinish

CSM0738

Service name:	Csm_KeyExchangeCalcSymKeyFinish	

Syntax:	`Csm_ReturnType Csm_KeyExchangeCalcSymKeyFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  Csm_SymKeyType* sharedKeyPtr`
	`)`	

Service ID[hex]:	0x3f	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	sharedKeyPtr	holds a pointer to the key which will hold the result of the key exchange
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the key exchange service.	

	If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_KeyExchangeCalcSymKeyFinish.

8.3.12 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.

8.3.12.1 Csm_SymKeyExtractStart

CSM0418

Service name:	Csm_SymKeyExtractStart	

Syntax:	`Csm_ReturnType Csm_SymKeyExtractStart(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x32	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	holds the identifier of the CSM module configuration which has to be used during the key extraction
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the symmetrical key extraction service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymKeyExtractStart.

8.3.12.2 Csm_SymKeyExtractUpdate

CSM0425

Service name:	Csm_SymKeyExtractUpdate	

Syntax:	`Csm_ReturnType Csm_SymKeyExtractUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`	

Service ID[hex]:	0x33	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	dataPtr	holds a pointer to the data which contains the key in a format which cannot be used directly by the CSM. From this data the key will be extracted in a CSM-conforming format
	dataLength	holds the length of the data in bytes
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the symmetrical key extraction service with input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_SymKeyExtractUpdate.

8.3.12.3 Csm_SymKeyExtractFinish

CSM0432

Service name:	Csm_SymKeyExtractFinish	

Syntax:	`Csm_ReturnType Csm_SymKeyExtractFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  Csm_SymKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x34	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	keyPtr	holds a pointer to a structure where the result (i.e. the symmetrical key) is stored in.
Parameters (out):	None	

Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the symmetrical key extraction service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_SymKeyExtractFinish.

8.3.13 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.

8.3.13.1 Csm_SymKeyWrapSymStart

CSM0739

Service name:	Csm_SymKeyWrapSymStart	

Syntax:	`Csm_ReturnType Csm_SymKeyWrapSymStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* keyPtr,`
	`  const Csm_SymKeyType* wrappingKeyPtr`
	`)`	

Service ID[hex]:	0x40	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non 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 key used for wrapping
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the symmetrical key wrapping service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymKeyWrapSymStart.

8.3.13.2 Csm_SymKeyWrapSymUpdate

CSM0740

Service name:	Csm_SymKeyWrapSymUpdate	

Syntax:	`Csm_ReturnType Csm_SymKeyWrapSymUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* dataPtr,`
	`  uint32* dataLengthPtr`
	`)`	

Service ID[hex]:	0x41	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
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 is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_SymKeyWrapSymUpdate.

8.3.13.3 Csm_SymKeyWrapSymFinish

CSM0741

Service name:	Csm_SymKeyWrapSymFinish	

Syntax:	`Csm_ReturnType Csm_SymKeyWrapSymFinish(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x42	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the symmetrical key wrapping service. If the	

	service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_SymKeyWrapSymFinish.

8.3.13.4 Csm_SymKeyWrapAsymStart

CSM0742

Service name:	Csm_SymKeyWrapAsymStart	

Syntax:	`Csm_ReturnType Csm_SymKeyWrapAsymStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_SymKeyType* keyPtr,`
	`  const Csm_AsymPublicKeyType* wrappingKeyPtr`
	`)`	

Service ID[hex]:	0x43	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the symmetrical key wrapping service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_SymKeyWrapAsymStart.

8.3.13.5 Csm_SymKeyWrapAsymUpdate

CSM0743

Service name:	Csm_SymKeyWrapAsymUpdate	

Syntax:	`Csm_ReturnType Csm_SymKeyWrapAsymUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* dataPtr,`
	`  uint32* dataLengthPtr`
	`)`	

Service ID[hex]:	0x44	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to

		be used during the operation.
Parameters (inout):	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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
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 is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_SymKeyWrapAsymUpdate.

8.3.13.6 Csm_SymKeyWrapAsymFinish

CSM0744

Service name:	Csm_SymKeyWrapAsymFinish	

Syntax:	`Csm_ReturnType Csm_SymKeyWrapAsymFinish(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x45	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the symmetrical key wrapping service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_SymKeyWrapAsymFinish.

8.3.14 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 these 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.

8.3.14.1 Csm_AsymPublicKeyExtractStart

CSM0436

Service name:	Csm_AsymPublicKeyExtractStart	

Syntax:	`Csm_ReturnType Csm_AsymPublicKeyExtractStart(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x35	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	hold the identifier of the CSM module configuration which has to be used during the key extraction.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the asymmetrical public key extraction service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymPublicKeyExtractStart.

8.3.14.2 Csm_AsymPublicKeyExtractUpdate

CSM0443

Service name:	Csm_AsymPublicKeyExtractUpdate

Syntax:	`Csm_ReturnType Csm_AsymPublicKeyExtractUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`

Service ID[hex]:	0x36
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)

Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	dataPtr	holds a pointer to the data which contains the key in a format which cannot be used directly by the CSM. From this data the key will be extracted in a CSM-conforming format
	dataLength	holds the length of the data in bytes
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to feed the asymmetrical public key extraction service with input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPublicKeyExtractUpdate.

8.3.14.3 Csm_AsymPublicKeyExtractFinish

CSM0450

Service name:	Csm_AsymPublicKeyExtractFinish	

Syntax:	`Csm_ReturnType Csm_AsymPublicKeyExtractFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  Csm_AsymPublicKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x37	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	keyPtr	holds a pointer to a structure where the result (i.e. the asymmetrical public key) is stored in
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the asymmetrical public key extraction service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPublicKeyExtractFinish.

8.3.14.4 Csm_AsymPrivateKeyExtractStart

CSM0680

Service name:	Csm_AsymPrivateKeyExtractStart	

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyExtractStart(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x38	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	hold the identifier of the CSM module configuration which has to be used during the key extraction.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the asymmetrical private key extraction service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyExtractStart.

8.3.14.5 Csm_AsymPrivateKeyExtractUpdate

CSM0682

Service name:	Csm_AsymPrivateKeyExtractUpdate	

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyExtractUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  const uint8* dataPtr,`
	`  uint32 dataLength`
	`)`	

Service ID[hex]:	0x39	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
	dataPtr	holds a pointer to the data which contains the key in a format which cannot be used directly by the CSM. From this data the key will be extracted in a CSM-conforming format
	dataLength	holds the length of the data in bytes
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy

Description:	This interface shall be used to feed the asymmetrical private key extraction service with input data. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyExtractUpdate.

8.3.14.6 Csm_AsymPrivateKeyExtractFinish

CSM0684

Service name:	Csm_AsymPrivateKeyExtractFinish	

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyExtractFinish(`
	`  Csm_ConfigIdType cfgId,`
	`  Csm_AsymPrivateKeyType* keyPtr`
	`)`	

Service ID[hex]:	0x3a	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	keyPtr	holds a pointer to a structure where the result (i.e. the asymmetrical private key) is stored in
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the asymmetrical private key extraction service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyExtractFinish.

8.3.15 Asymmetric key wrapping interface

Asymmetric 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.

8.3.15.1 Csm_AsymPrivateKeyWrapSymStart

CSM0745

Service name:	Csm_AsymPrivateKeyWrapSymStart
Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyWrapSymStart(`

	`Csm_ConfigIdType cfgId,`
	`const Csm_AsymPrivateKeyType* keyPtr,`
	`const Csm_SymKeyType* wrappingKeyPtr`
	)	

Service ID[hex]:	0x46	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non 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 key used for wrapping
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the asymmetrical key wrapping service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyWrapSymStart.

8.3.15.2 Csm_AsymPrivateKeyWrapSymUpdate

CSM0746

Service name:	Csm_AsymPrivateKeyWrapSymUpdate	

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyWrapSymUpdate(`
	`Csm_ConfigIdType cfgId,`
	`uint8* dataPtr,`
	`uint32* dataLengthPtr`
	)	

Service ID[hex]:	0x47	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	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:	Csm_ReturnType	CSM_E_OK: request successful

		CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
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 is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyWrapSymUpdate.

8.3.15.3 Csm_AsymPrivateKeyWrapSymFinish

CSM0747

Service name:	Csm_AsymPrivateKeyWrapSymFinish	

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyWrapSymFinish(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0xxx	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the asymmetrical key wrapping service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyWrapSymFinish.

8.3.15.4 Csm_AsymPrivateKeyWrapAsymStart

CSM0748

Service name:	Csm_AsymPrivateKeyWrapAsymStart

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyWrapAsymStart(`
	`  Csm_ConfigIdType cfgId,`
	`  const Csm_AsymPrivateKeyType* keyPtr,`
	`  const Csm_AsymPublicKeyType* wrappingKeyPtr`
	`)`

Service ID[hex]:	0x49
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)
Reentrancy:	Non 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):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to initialize the asymmetrical key wrapping service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".	

Regarding error detection, the requirements CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyWrapAsymStart.

8.3.15.5 Csm_AsymPrivateKeyWrapAsymUpdate

CSM0749

Service name:	Csm_AsymPrivateKeyWrapAsymUpdate	

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyWrapAsymUpdate(`
	`  Csm_ConfigIdType cfgId,`
	`  uint8* dataPtr,`
	`  uint32* dataLengthPtr`
	`)`	

Service ID[hex]:	0x4a	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	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:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
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 is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the	

	primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyWrapAsymUpdate.

8.3.15.6 Csm_AsymPrivateKeyWrapAsymFinish

CSM0750

Service name:	Csm_AsymPrivateKeyWrapAsymFinish	

Syntax:	`Csm_ReturnType Csm_AsymPrivateKeyWrapAsymFinish(`
	`  Csm_ConfigIdType cfgId`
	`)`	

Service ID[hex]:	0x4b	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	cfgId	Holds the identifier of the CSM module configuration that has to be used during the operation.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description:	This interface shall be used to finish the asymmetrical key wrapping service. If the service state is "idle", the function has to return with "CSM_E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information 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 CSM0488 and CSM0489 are applicable to the function Csm_AsymPrivateKeyWrapAsymFinish.

8.4 Dependencies to cryptographic library API functions

8.4.1 Types for the Cryptographic Primitives

8.4.1.1 Cry_<Primitive>ConfigType

CSM0544

Name:	`Cry_<Primitive>ConfigType`		
Type:	`Structure`		
Element:	`void`	`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

CSM0461

For every API function of a cryptographic service, the corresponding cryptographic primitive shall contain a corresponding function.

CSM0505

The implementation of the basic cryptographic routines shall be synchronous or asynchronous, depending on the configuration of the CSM.

8.4.2.1 Cry_<Primitive>Start

CSM0701

Service name:	Cry_<Primitive>Start	

Syntax:	`Csm_ReturnType Cry_<Primitive>Start(`
	`  <type> <xxx>`
	`)`	

Service ID[hex]:	--	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Csm_<Service>Start(), with the exception of the argument cfgId. This argument is of type “Csm_ConfigIdType“ in Csm_<Service>Start(). In Cry_<Primitive>Start the argument cfgId shall be replaced by an argument cfgPtr of type “const void *“.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	The return values shall be identical to those of the corresponding function Csm_<Service>Start().
Description:	Synchronous: This function shall initialize the computation of the cryptographic primitive, so that the primitive is able to process input data. Asynchronous: This function shall store the information given in the arguments, so that Cry_<Primitive>MainFunction() can process the initialisation.	

CSM0732

The API "Cry_<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 "Cry_<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 Csm_<Service>ConfigType one element is a function pointer to this API.

8.4.2.2 Cry_<Primitive>Update

CSM0702

Service name:	Cry_<Primitive>Update
Syntax:	`Csm_ReturnType Cry_<Primitive>Update(`

	<type> <xxx>)	
Service ID[hex]:	--	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Csm_<Service>Update().
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	The return values shall be identical to those of the corresponding function Csm_<Service>Update().
Description:	Synchronous: This function shall process a chunk of the given input data with the algorithm of the cryptographic primitive. Asynchronous: This function shall store the information given in the arguments, so that Cry_<Primitive>MainFunction() can process the input data. If the primitive is not ready to process the current request (e.g. because the processing of another Cry_<Primitive>Update has not yet finished), the function shall return with "CSM_E_BUSY".	

8.4.2.3 Cry_<Primitive>Finish

CSM0703

Service name:	Cry_<Primitive>Finish	
Syntax:	Csm_ReturnType Cry_<Primitive>Finish(<type> <xxx>)	
Service ID[hex]:	--	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Csm_<Service>Finish().
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	The return values shall be identical to those of the corresponding function Csm_<Service>Finish().
Description:	Synchronous: This function shall finish the computation of the cryptographic primitive and store the result into the memory location given. Asynchronous: This function shall store the information given in the arguments, so that Cry_<Primitive>MainFunction() can finish the computation and store the result in the memory location given. If the primitive is not ready to process the current request (e.g. because the processing of a Cry_<Primitive>Update has not yet finished), the function has to return with "CSM_E_BUSY".	

8.4.2.4 Cry_<Primitive>

CSM0704

Service name:	Cry_<Primitive>	

Syntax:	`Csm_ReturnType Cry_<Primitive>(`
	`  <type> <xxx>`
	`)`	

Service ID[hex]:	--	
Sync/Async:	Sync or Async, dependent on configuration (CSM0557_Conf)	
Reentrancy:	Non Reentrant	
Parameters (in):	<xxx>	The arguments <xxx> shall be identical to the arguments of the corresponding function Csm_<Service>(), with the exception of the argument cfgId. This argument is of type "Csm_ConfigIdType" in Csm_<Service>(). In Cry_<Primitive> the argument cfgId shall be replaced by an argument cfgPtr of type "const void *".
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Csm_ReturnType	The return values shall be identical to those of the corresponding function Csm_<Service>().
Description:	Synchronous: This function shall process the cryptographic primitive with the given input data and store the result in the memory location given. Asynchronous: This function shall prepare the computation of the cryptographic primitive, so that the primitive main function is able to process the input data and return the result.	

CSM0733
The API "Cry_<Primitive>" has a parameter "cfgPtr" of type "const void *".
When calling this API, the parameter "cfgPtr" shall point to a constant variable of type "Cry_<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 Csm_<Service>ConfigType one element is a function pointer to this API.

8.4.2.5 Cry_<Primitive>MainFunction

CSM0773

Service name:	Cry_<Primitive>MainFunction

Syntax:	`void Cry_<Primitive>MainFunction(`
	`  void`
	`)`

Service ID[hex]:	--
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The calculation of the cryptographic functions shall be done in the Cry_<Primitive>MainFunction(). When the main function has completely processed the cryptographic functions demanded by Cry_<Primitive>Start() or Cry_<Primitive>Update(), the corresponding callback Csm_<Service>CallbackNotification() must be called with the correct return value. When the main function has completely processed the cryptographic functions

	demanded by Cry_<Primitive>Finish(), the callback Csm_<Service>CallbackNotification() must be called with the correct return value and then the callback Csm_<Service>ServiceFinishNotification() must be called.

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 `Cry_<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 `Csm<Service>InitConfiguration` of the service configuration has to be set to the C-language symbol of the primitive configuration.

Variants of CRY 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.

8.5 Call-back notifications

8.5.1 CRY callback notifications

CSM0454
When the cryptographic library has to change the main state machine of the CSM, this shall be done by using the following functions:

8.5.1.1 Csm_<Service>CallbackNotification

CSM0455

Service name:	Csm_<Service>CallbackNotification	
Syntax:	`void Csm_<Service>CallbackNotification( Csm_ReturnType Result )`	
Service ID[hex]:	--	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Result	Contains the result of a cryptographic operation. CSM_E_OK: request successful CSM_E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy CSM_E_SMALL_BUFFER: the provided buffer is too small to store the result CSM_E_ENTROPY_EXHAUSTION: request failed, entropy of randum number generator is exhausted.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	

Description:	This function shall call the callback function as given in the configuration of the service <Service> with the argument given by "Result".

8.5.1.2 Csm_<Service>ServiceFinishNotification

CSM0457

Service name:	Csm_<Service>ServiceFinishNotification

Syntax:	`void Csm_<Service>ServiceFinishNotification(`
	`  void`
	`)`

Service ID[hex]:	--
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This function shall set the state of the service <Service> to "idle".

8.5.2 User callback notifications

CSM0535

User callback notifications are configured in the structure Csm_<Service>ConfigType (see CSM0074).

8.6 Scheduled functions

CSM0476

These functions are directly called by Basic Software Scheduler.

CSM0477

The following functions shall have no return value and no parameter.

CSM0536

All functions shall be non reentrant.

8.6.1 Csm_MainFunction

CSM0479

Service name:	Csm_MainFunction

Syntax:	`void Csm_MainFunction(`
	`  void`
	`)`

Service ID[hex]:	0x01
Timing:	FIXED_CYCLIC
Description:	API to be called cyclically to process the requested services.

CSM0480

This function shall perform the processing of the CSM module jobs.

CSM0469

If a cryptographic service is active, the Csm_MainFunction() shall call the corresponding Cry_<Primitive>MainFunction() to calculate the cryptographic primitive.

CSM0481

This function has to be called cyclically in every case.

CSM0483

If no further job processing is possible, the Csm_MainFunction shall return immediately.

8.7 Interfaces to standard software modules

CSM0484

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

CSM0485

The CSM module shall use an AUTOSAR Det module for development error notification.

CSM0486

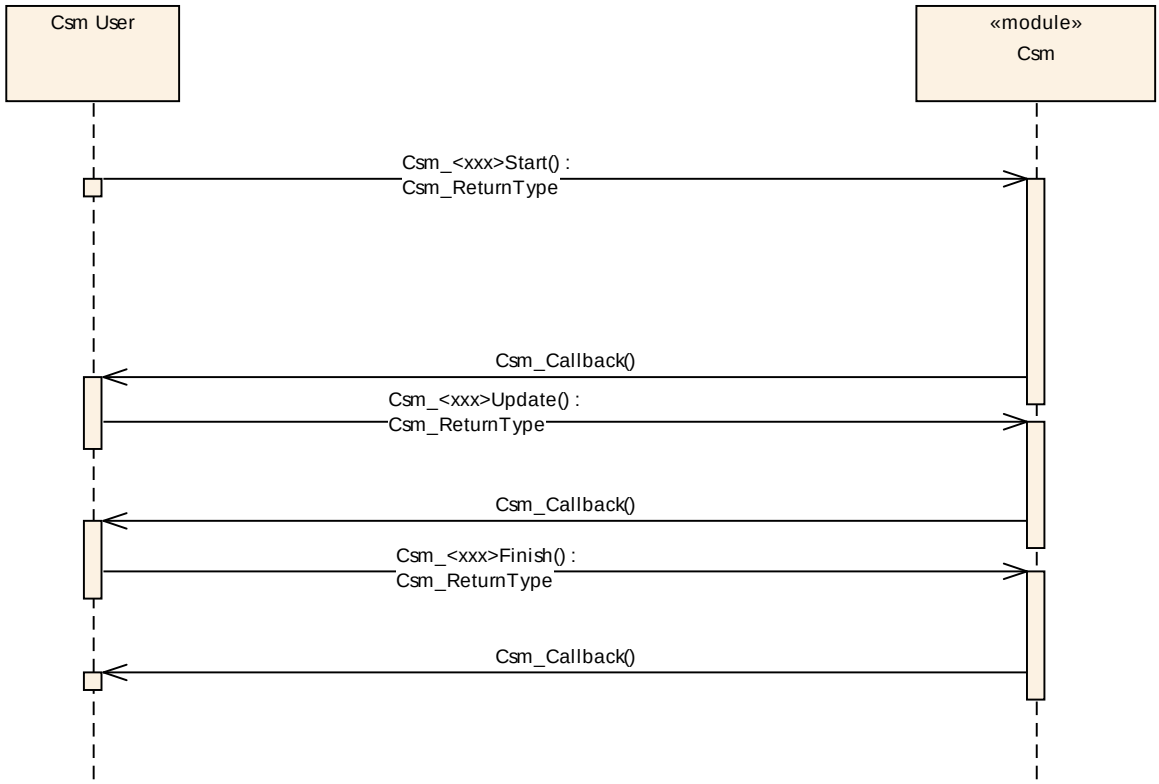
The CSM module shall use an AUTOSAR Dem module to report errors to the DEM.

9 Sequence diagrams

The following sequence diagrams concentrate on the interaction between the CSM module and software components respectively the ECU state manager.

9.1 Asynchronous calls

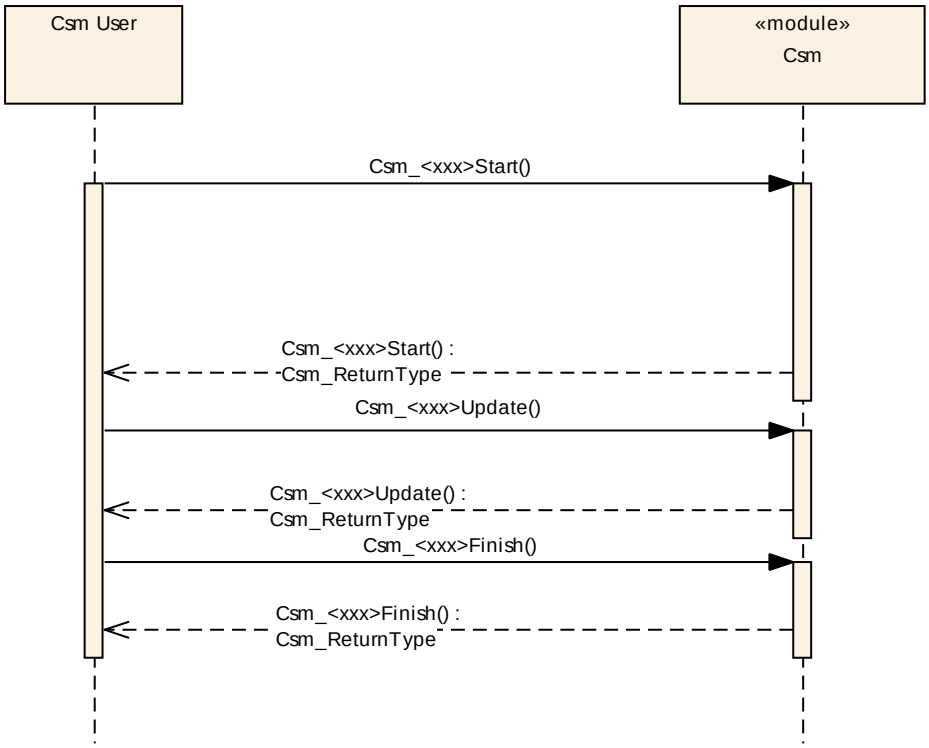
The following diagram (Sequence diagram for asynchronous call) shows a sample sequence of function calls for a request which is performed asynchronously. The result of the asynchronous function can be accessed after an asynchronous notification (invocation of the configured callback function).



Sequence diagram for asynchronous call with callback

9.2 Synchronous calls

The following diagram (Sequence diagram for synchronous calls) shows a sample sequence of function calls with the scheduler for a request which is performed synchronously.



Sequence diagram for synchronous call

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 CSM.

Chapter 10.3 specifies published information of the module CSM.

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 [6]
- 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 Csm

Module Name	*Csm*
Module Description	Configuration of the Csm (CryptoServiceManager) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CsmAsymDecrypt	0..1	Container for incorporation of AsymDecrypt primitives.
CsmAsymEncrypt	0..1	Container for incorporation of AsymEncrypt primitives.
CsmAsymPrivateKeyExtract	0..1	Container for incorporation of AsymPrivateKeyExtract primitives.
CsmAsymPrivateKeyWrapAsym	0..1	Container for incorporation of AsymPrivateKeyWrapSym primitives.
CsmAsymPrivateKeyWrapSym	0..1	Container for incorporation of AsymPrivateKeyWrapSym primitives.
CsmAsymPublicKeyExtract	0..1	Container for incorporation of AsymPublicKeyExtract primitives.
CsmChecksum	0..1	Container for incorporation of Checksum primitives.
CsmDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
CsmGeneral	1	Container for common configuration options.
CsmHash	0..1	Container for incorporation of Hash primitives.
CsmKeyDerive	0..1	Container for incorporation of KeyDerive primitives.
CsmKeyDeriveSymKey	0..1	Container for incorporation of CsmKeyDeriveSymKey primitives.
CsmKeyExchangeCalcPubVal	0..1	Container for incorporation of KeyExchangeCalcPubVal primitives.
CsmKeyExchangeCalcSecret	0..1	Container for incorporation of KeyExchangeCalcSecret primitives.
CsmKeyExchangeCalcSymKey	0..1	Container for incorporation of KeyExchangeCalcSymKey primitives.
CsmMacGenerate	0..1	Container for incorporation of MacGenerate primitives.
CsmMacVerify	0..1	Container for incorporation of MacVerify primitives.
CsmRandomGenerate	0..1	Container for incorporation of RandomGenerate primitives.

CsmRandomSeed	0..1	Container for incorporation of RandomSeed primitives.
CsmSignatureGenerate	0..1	Container for incorporation of SignatureGenerate primitives.
CsmSignatureVerify	0..1	Container for incorporation of SignatureVerify primitives.
CsmSymBlockDecrypt	0..1	Container for incorporation of SymBlockDecrypt primitives.
CsmSymBlockEncrypt	0..1	Container for incorporation of SymBlockEncrypt primitives.
CsmSymDecrypt	0..1	Container for incorporation of SymDecrypt primitives.
CsmSymEncrypt	0..1	Container for incorporation of SymEncrypt primitives.
CsmSymKeyExtract	0..1	Container for incorporation of SymKeyExtract primitives.
CsmSymKeyWrapAsym	0..1	Container for incorporation of SymKeyWrapSym primitives.
CsmSymKeyWrapSym	0..1	Container for incorporation of SymKeyWrapSym primitives.

10.2.3 CsmGeneral

SWS Item	**CSM0554_Conf :**
Container Name	CsmGeneral
Description	Container for common configuration options.
Configuration Parameters	

SWS Item	**CSM0555_Conf :**		
Name	CsmDevErrorDetect		
Description	Pre-processor switch to enable and disable development error detection. True: Development error detection enabled. False: Development error detection disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0729_Conf :**		
Name	CsmMaxAlignScalarType		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0558_Conf :**
Name	CsmMaximumBlockingTime
Description	If interruption is turned on with the configuration option CsmUseInterruption, this option configures the maximum time in microseconds the main function shall be allowed to

	run before it must interrupt itself. The lowest allowed value for the option is implementation dependent.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0556_Conf :**		
Name	CsmUseInterruption		
Description	Pre-processor switch to enable and disable interruption of job processing. True: Interruption of job processing enabled False: Interruption of job processing disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0557_Conf :**		
Name	CsmUseSyncJobProcessing		
Description	Pre-processor switch to enable and disable synchronous job processing. True: synchronous job processing enabled False: synchronous job processing disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0708_Conf :**		
Name	CsmVersionInfoApi		
Description	Pre-processor switch to enable and disable availability of the API Csm_GetVersionInfo(). True: API Csm_GetVersionInfo() is available. False: API Csm_GetVersionInfo() is not available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.4 CsmHash

SWS Item	**CSM0559_Conf :**
Container Name	CsmHash
Description	Container for incorporation of Hash primitives.
Configuration Parameters	

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

10.2.5 CsmHashConfig

SWS Item	**CSM0560_Conf :**
Container Name	CsmHashConfig
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	**CSM0561_Conf :**		
Name	CsmCallbackHash		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0563_Conf :**		
Name	CsmHashInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**Csm0562_Conf :**
Name	CsmHashPrimitiveName

Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.6 CsmMacGenerate

SWS Item	**CSM0635_Conf :**
Container Name	CsmMacGenerate
Description	Container for incorporation of MacGenerate primitives.
Configuration Parameters	

SWS Item	**CSM0709_Conf :**		
Name	CsmMacGenerateMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a MAC generation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.7 CsmMacGenerateConfig

SWS Item	**CSM0564_Conf :**
Container Name	CsmMacGenerateConfig
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	**CSM0565_Conf :**
Name	CsmCallbackMacGenerate

Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0567_Conf :**		
Name	CsmMacGenerateInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0566_Conf :**		
Name	CsmMacGeneratePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.8 CsmMacVerify

SWS Item	**CSM0636_Conf :**
Container Name	CsmMacVerify
Description	Container for incorporation of MacVerify primitives.
Configuration Parameters	

SWS Item	**CSM0710_Conf :**
Name	CsmMacVerifyMaxKeySize

Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a MAC verification.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.9 CsmMacVerifyConfig

SWS Item	**CSM0568_Conf :**
Container Name	CsmMacVerifyConfig
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	**CSM0569_Conf :**		
Name	CsmCallbackMacVerify		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0571_Conf :**		
Name	CsmMacVerifyInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0570_Conf :		
Name	CsmMacVerifyPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.10 CsmRandomSeed

SWS Item	CSM0641_Conf :
Container Name	CsmRandomSeed
Description	Container for incorporation of RandomSeed primitives.
Configuration Parameters	

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

10.2.11 CsmRandomSeedConfig

SWS Item	CSM0642_Conf :
Container Name	CsmRandomSeedConfig
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	CSM0643_Conf :		
Name	CsmCallbackRandomSeed		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	

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

SWS Item	**CSM0645_Conf :**		
Name	CsmRandomSeedInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0644_Conf :**		
Name	CsmRandomSeedPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.12 CsmRandomGenerate

SWS Item	**CSM0620 :**
Container Name	CsmRandomGenerate
Description	Container for incorporation of RandomGenerate primitives.
Configuration Parameters	

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

10.2.13 CsmRandomGenerateConfig

SWS Item	**CSM0637_Conf :**

Container Name	CsmRandomGenerateConfig
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	**CSM0638_Conf :**		
Name	CsmCallbackRandomGenerate		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0640_Conf :**		
Name	CsmRandomGenerateInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0639_Conf :**		
Name	CsmRandomGeneratePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.14 CsmSymBlockEncrypt

SWS Item	**CSM0621_Conf :**
Container Name	CsmSymBlockEncrypt
Description	Container for incorporation of SymBlockEncrypt primitives.
Configuration Parameters	

SWS Item	**CSM0711_Conf :**		
Name	CsmSymBlockEncryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a symmetrical block encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.15 CsmSymBlockEncryptConfig

SWS Item	**CSM0572_Conf :**
Container Name	CsmSymBlockEncryptConfig
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	**CSM0573_Conf :**		
Name	CsmCallbackSymBlockEncrypt		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0575_Conf :**
Name	CsmSymBlockEncryptInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1

Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0574_Conf :**		
Name	CsmSymBlockEncryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.16 CsmSymBlockDecrypt

SWS Item	**CSM0622_Conf :**
Container Name	CsmSymBlockDecrypt
Description	Container for incorporation of SymBlockDecrypt primitives.
Configuration Parameters	

SWS Item	**CSM0712_Conf :**		
Name	CsmSymBlockDecryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a symmetrical block decryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.17 CsmSymBlockDecryptConfig

SWS Item	**CSM0576_Conf :**
Container Name	CsmSymBlockDecryptConfig
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	**CSM0577_Conf :**		
Name	CsmCallbackSymBlockDecrypt		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0579_Conf :**		
Name	CsmSymBlockDecryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0578_Conf :**		
Name	CsmSymBlockDecryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.18 CsmSymEncrypt

SWS Item	**CSM0623_Conf :**
Container Name	CsmSymEncrypt
Description	Container for incorporation of SymEncrypt primitives.
Configuration Parameters	

SWS Item	**CSM0713_Conf :**		
Name	CsmSymEncryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a symmetrical encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.19 CsmSymEncryptConfig

SWS Item	**CSM0580_Conf :**
Container Name	CsmSymEncryptConfig
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	**CSM0581_Conf :**		
Name	CsmCallbackSymEncrypt		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0583_Conf :**
Name	CsmSymBlockEncryptInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1

Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0582_Conf :**		
Name	CsmSymEncryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.20 CsmSymDecrypt

SWS Item	**CSM0624_Conf :**
Container Name	CsmSymDecrypt
Description	Container for incorporation of SymDecrypt primitives
Configuration Parameters	

SWS Item	**CSM0714_Conf :**		
Name	CsmSymDecryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a symmetrical decryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.21 CsmSymDecryptConfig

SWS Item	**CSM0584_Conf :**
Container Name	CsmSymDecryptConfig
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	**CSM0585_Conf :**		
Name	CsmCallbackSymDecrypt		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0587_Conf :**		
Name	CsmSymDecryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0586_Conf :**		
Name	CsmSymDecryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.22 CsmAsymEncrypt

SWS Item	**CSM0625_Conf :**		
Container Name	CsmAsymEncrypt		
Description	Container for incorporation of AsymEncrypt primitives.		
Configuration Parameters			

SWS Item	**CSM0715_Conf :**		
Name	CsmAsymEncryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement an asymmetrical encryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.23 CsmAsymEncryptConfig

SWS Item	**CSM0588_Conf :**
Container Name	CsmAsymEncryptConfig
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	**CSM0591_Conf :**		
Name	CsmAsymEncryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0590_Conf :**
Name	CsmAsymEncryptPrimitiveName
Description	Name of the cryptographic primitive to use.
Multiplicity	1

Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0589_Conf :**		
Name	CsmCallbackAsymEncrypt		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.24 CsmAsymDecrypt

SWS Item	**CSM0626_Conf :**
Container Name	CsmAsymDecrypt
Description	Container for incorporation of AsymDecrypt primitives.
Configuration Parameters	

SWS Item	**CSM0716_Conf :**		
Name	CsmAsymDecryptMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement an asymmetrical decryption.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.25 CsmAsymDecryptConfig

SWS Item	**CSM0592_Conf :**
Container Name	CsmAsymDecryptConfig
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	**CSM0595_Conf :**		
Name	CsmAsymDecryptInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0594_Conf :**		
Name	CsmAsymDecryptPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0593_Conf :**		
Name	CsmCallbackAsymDecrypt		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.26 CsmSignatureGenerate

SWS Item	**CSM0627_Conf :**
Container Name	CsmSignatureGenerate
Description	Container for incorporation of SignatureGenerate primitives
Configuration Parameters	

SWS Item	**CSM0717_Conf :**		
Name	CsmSignatureGenerateMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a signature generation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.27 CsmSignatureGenerateConfig

SWS Item	**CSM0596_Conf :**
Container Name	CsmSignatureGenerateConfig
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	**CSM0597_Conf :**		
Name	CsmCallbackSignatureGenerate		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0599_Conf :**
Name	CsmSignatureGenerateInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1

Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0598_Conf :**		
Name	CsmSignatureGeneratePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.28 CsmSignatureVerify

SWS Item	**CSM0628_Conf :**
Container Name	CsmSignatureVerify
Description	Container for incorporation of SignatureVerify primitives.
Configuration Parameters	

SWS Item	**CSM0718_Conf :**		
Name	CsmSignatureVerifyMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a signature verification.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.29 CsmSignatureVerifyConfig

SWS Item	**CSM0600_Conf :**
Container Name	CsmSignatureVerifyConfig
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	**CSM0601_Conf :**		
Name	CsmCallbackSignatureVerify		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0603_Conf :**		
Name	CsmSignatureVerifyInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0602_Conf :**		
Name	CsmSignatureVerifyPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.30 CsmChecksum

SWS Item	**CSM0629_Conf :**
Container Name	CsmChecksum
Description	Container for incorporation of Checksum primitives.
Configuration Parameters	

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

10.2.31 CsmChecksumConfig

SWS Item	**CSM0604_Conf :**
Container Name	CsmChecksumConfig
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	**CSM0605_Conf :**		
Name	CsmCallbackChecksum		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0607_Conf :**		
Name	CsmChecksumInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0606_Conf :**
Name	CsmChecksumPrimitiveName
Description	Name of the cryptographic primitive to use.

Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.32 CsmKeyDerive

SWS Item	**CSM0630_Conf :**
Container Name	CsmKeyDerive
Description	Container for incorporation of KeyDerive primitives.
Configuration Parameters	

SWS Item	**CSM0719_Conf :**		
Name	CsmKeyDeriveMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a key derivation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.33 CsmKeyDeriveConfig

SWS Item	**CSM0608_Conf :**
Container Name	CsmKeyDeriveConfig
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	**CSM0609_Conf :**
Name	CsmCallbackKeyDerive

Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0611_Conf :**		
Name	CsmKeyDeriveInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0610_Conf :**		
Name	CsmKeyDerivePrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.34 CsmKeyExchangeCalcPubVal

SWS Item	**CSM0631_Conf :**
Container Name	CsmKeyExchangeCalcPubVal
Description	Container for incorporation of KeyExchangeCalcPubVal primitives.
Configuration Parameters	

SWS Item	**CSM0720_Conf :**
Name	CsmKeyExchangeCalcPubValMaxBaseTypeSize

Description	The maximum length, in bytes, of all base types used in all CRY primitives which implement a public value calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0721_Conf :**		
Name	CsmKeyExchangeCalcPubValMaxPrivateTypeSize		
Description	The maximum length, in bytes, of all private information types used in all CRY primitives which implement a public value calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.35 CsmKeyExchangeCalcPubValConfig

SWS Item	**CSM0612_Conf :**
Container Name	CsmKeyExchangeCalcPubValConfig
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	**CSM0613_Conf :**		
Name	CsmCallbackKeyExchangeCalcPubVal		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0615_Conf :**

Name	CsmKeyExchangeCalcPubValInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0614_Conf :**		
Name	CsmKeyExchangeCalcPubValPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.36 CsmKeyExchangeCalcSecret

SWS Item	**CSM0632_Conf :**
Container Name	CsmKeyExchangeCalcSecret
Description	Container for incorporation of KeyExchangeCalcSecret primitives.
Configuration Parameters	

SWS Item	**CSM0722_Conf :**		
Name	CsmKeyExchangeCalcSecretMaxBaseTypeSize		
Description	The maximum length, in bytes, of all base types used in all CRY primitives which implement a shared secret calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0723_Conf :**
Name	CsmKeyExchangeCalcSecretMaxPrivateTypeSize

Description	The maximum length, in bytes, of all private information types used in all CRY primitives which implement a shared secret calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.37 CsmKeyExchangeCalcSecretConfig

SWS Item	**CSM0616_Conf :**
Container Name	CsmKeyExchangeCalcSecretConfig
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	**CSM0617_Conf :**		
Name	CsmCallbackKeyExchangeCalcSecret		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0545_Conf :**		
Name	CsmKeyExchangeCalcSecretInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0618_Conf :**		
Name	CsmKeyExchangeCalcSecretPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.38 CsmSymKeyExtract

SWS Item	**CSM0633_Conf :**
Container Name	CsmSymKeyExtract
Description	Container for incorporation of SymKeyExtract primitives.
Configuration Parameters	

SWS Item	**CSM0724_Conf :**		
Name	CsmSymKeyExtractMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a symmetrical key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.39 CsmSymKeyExtractConfig

SWS Item	**CSM0546_Conf :**
Container Name	CsmSymKeyExtractConfig
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	CSM0547_Conf :		
Name	CsmCallbackSymKeyExtract		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0549_Conf :		
Name	CsmSymKeyExtractInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0548_Conf :		
Name	CsmSymKeyExtractPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.40 CsmAsymPublicKeyExtract

SWS Item	CSM0634_Conf :
Container Name	CsmAsymPublicKeyExtract
Description	Container for incorporation of AsymPublicKeyExtract primitives.
Configuration Parameters	

SWS Item	**CSM0725_Conf :**		
Name	CsmAsymPublicKeyExtractMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement an asymmetrical public key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.41 CsmAsymPublicKeyExtractConfig

SWS Item	**CSM0550_Conf :**
Container Name	CsmAsymPublicKeyExtractConfig
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	**CSM0553_Conf :**		
Name	CsmAsymPublicKeyExtractInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0552_Conf :**		
Name	CsmAsymPublicKeyExtractPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: module

SWS Item	**CSM0551_Conf :**		
Name	CsmCallbackAsymPublicKeyExtract		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.42 CsmAsymPrivateKeyExtract

SWS Item	**CSM0686_Conf :**
Container Name	CsmAsymPrivateKeyExtract
Description	Container for incorporation of AsymPrivateKeyExtract primitives.
Configuration Parameters	

SWS Item	**CSM0726_Conf :**		
Name	CsmAsymPrivateKeyExtractMaxKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement an asymmetrical private key extraction.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.43 CsmAsymPrivateKeyExtractConfig

SWS Item	**CSM0687_Conf :**
Container Name	CsmAsymPrivateKeyExtractConfig
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	CSM0690_Conf :		
Name	CsmAsymPrivateKeyExtractInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0689_Conf :		
Name	CsmAsymPrivateKeyExtractPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0688_Conf :		
Name	CsmCallbackAsymPrivateKeyExtract		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.44 CsmKeyExchangeCalcSymKey

SWS Item	CSM0732_Conf :
Container Name	CsmKeyExchangeCalcSymKey
Description	Container for incorporation of KeyExchangeCalcSymKey primitives.

Configuration Parameters

SWS Item	**CSM0738_Conf :**		
Name	CsmKeyExchangeCalcSymKeyMaxBaseTypeSize		
Description	The maximum length, in bytes, of all base types used in all CRY primitives which implement a symmetrical key calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0737_Conf :**		
Name	CsmKeyExchangeCalcSymKeyMaxPrivateTypeSize		
Description	The maximum length, in bytes, of all private information types used in all CRY primitives which implement a symmetrical key calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0739_Conf :**		
Name	CsmKeyExchangeCalcSymKeyMaxSymKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a symmetrical key calculation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.45 CsmKeyExchangeCalcSymKeyConfig

SWS Item	**CSM0736_Conf :**
Container Name	CsmKeyExchangeCalcSymKeyConfig
Description	Container for configuration of service KeyExchangeCalcSymKey. The

	container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	**CSM0733_Conf :**		
Name	CsmCallbackKeyExchangeCalcSymKey		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0735_Conf :**		
Name	CsmKeyExchangeCalcSymKeyInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0734_Conf :**		
Name	CsmKeyExchangeCalcSymKeyPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.46 CsmAsymPrivateKeyWrapSym

SWS Item	**CSM0752_Conf :**
Container Name	CsmAsymPrivateKeyWrapSym
Description	Container for incorporation of AsymPrivateKeyWrapSym primitives.

Configuration Parameters			
SWS Item	**CSM0758_Conf :**		
Name	CsmAsymPrivateKeyWrapSymMaxPrivKeySize		
Description	The maximum length, in bytes, of all private information types used in all CRY primitives which implement an asymetric private key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0757_Conf :		
Name	CsmAsymPrivateKeyWrapSymMaxSymKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement an asymetrical private key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.47 CsmAsymPrivateKeyWrapSymConfig

SWS Item	CSM0753_Conf :
Container Name	CsmAsymPrivateKeyWrapSymConfig
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	CSM0756_Conf :
Name	CsmAsymPrivateKeyWrapSymInitConfiguration
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.
Multiplicity	1
Type	EcucStringParamDef
Default value	--
maxLength	--
minLength	--
regularExpression	--

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0755_Conf :**		
Name	CsmAsymPrivateKeyWrapSymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0754_Conf :**		
Name	CsmCallbackAsymPrivateKeyWrapSym		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.48 CsmAsymPrivateKeyWrapAsym

SWS Item	**CSM0759_Conf :**
Container Name	CsmAsymPrivateKeyWrapAsym
Description	Container for incorporation of AsymPrivateKeyWrapSym primitives.
Configuration Parameters	

SWS Item	**CSM0765_Conf :**		
Name	CsmAsymPrivateKeyWrapAsymMaxPrivKeySize		
Description	The maximum length, in bytes, of all private key types used in all CRY primitives which implement an asymmetrical private key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0764_Conf :**		
Name	CsmAsymPrivateKeyWrapAsymMaxPubKeySize		
Description	The maximum length, in bytes, of all public key types used in all CRY primitives which implement an asymmetrical private key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CsmAsymPrivateKeyWrapAsymConfig	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 CsmAsymPrivateKeyWrapAsymConfig

SWS Item	**CSM0760_Conf :**
Container Name	CsmAsymPrivateKeyWrapAsymConfig
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	**CSM0763_Conf :**		
Name	CsmAsymPrivateKeyWrapAsymInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0762_Conf :**		
Name	CsmAsymPrivateKeyWrapAsymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants

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

SWS Item	**CSM0761_Conf :**		
Name	CsmCallbackAsymPrivateKeyWrapAsym		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.50 CsmSymKeyWrapSym

SWS Item	**CSM0740_Conf :**
Container Name	CsmSymKeyWrapSym
Description	Container for incorporation of SymKeyWrapSym primitives.
Configuration Parameters	

SWS Item	**CSM0745_Conf :**		
Name	CsmSymKeyWrapSymMaxSymKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement an symmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CsmSymKeyWrapSymConfig	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 CsmSymKeyWrapSymConfig

SWS Item	**CSM0741_Conf :**
Container Name	CsmSymKeyWrapSymConfig
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	**CSM0742_Conf :**		
Name	CsmCallbackSymKeyWrapSym		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0744_Conf :**		
Name	CsmSymKeyWrapSymInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0743_Conf :**		
Name	CsmSymKeyWrapSymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.52 CsmSymKeyWrapAsym

SWS Item	**CSM0746_Conf :**
Container Name	CsmSymKeyWrapAsym
Description	Container for incorporation of SymKeyWrapSym primitives.
Configuration Parameters	

SWS Item	CSM0766_Conf :		
Name	CsmSymKeyWrapAsymMaxPubKeySize		
Description	The maximum length, in bytes, of all public key types used in all CRY primitives which implement an asymmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0748_Conf :		
Name	CsmSymKeyWrapAsymMaxSymKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement an asymmetrical key wrapping.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

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

10.2.53 CsmSymKeyWrapAsymConfig

SWS Item	CSM0747_Conf :
Container Name	CsmSymKeyWrapAsymConfig
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	CSM0749_Conf :		
Name	CsmCallbackSymKeyWrapAsym		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: module

SWS Item	**CSM0751_Conf :**		
Name	CsmSymKeyWrapAsymInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CSM0750_Conf :**		
Name	CsmSymKeyWrapAsymPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.54 CsmKeyDeriveSymKey

SWS Item	**CSM0767_Conf :**
Container Name	CsmKeyDeriveSymKey
Description	Container for incorporation of CsmKeyDeriveSymKey primitives.
Configuration Parameters	

SWS Item	**CSM0769_Conf :**		
Name	CsmKeyDeriveSymKeyMaxSymKeySize		
Description	The maximum, in bytes, of all key lengths used in all CRY primitives which implement a key derivation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

Included Containers

Container Name	Multiplicity	Scope / Dependency
CsmKeyDeriveSymKeyConfig	0..32	Container for configuration of service CsmKeyDeriveSymKey. The container name serves as a symbolic name for the identifier of a service configuration.

10.2.55 CsmKeyDeriveSymKeyConfig

SWS Item	CSM0768_Conf :
Container Name	CsmKeyDeriveSymKeyConfig
Description	Container for configuration of service CsmKeyDeriveSymKey. The container name serves as a symbolic name for the identifier of a service configuration.
Configuration Parameters	

SWS Item	CSM0770_Conf :		
Name	CsmCallbackKeyDeriveSymKey		
Description	Callback function to be called if service has finished.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0772_Conf :		
Name	CsmKeyDeriveSymKeyInitConfiguration		
Description	Name of a C symbol which contains the configuration of the underlying cryptographic primitive.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	CSM0771_Conf :		
Name	CsmKeyDeriveSymKeyPrimitiveName		
Description	Name of the cryptographic primitive to use.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	

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

No Included Containers

10.2.56 CsmDemEventParameterRefs

SWS Item	**CSM0730_Conf :**
Container Name	CsmDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**CSM0731_Conf :**		
Name	CSM_E_INIT_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "Initialization of CSM module failed" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.3 Published Information

[CSM001_PI] The standardized common published parameters as required by BSW00402 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].

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

11 AUTOSAR Service implemented by the CSM module

11.1 Scope of this Chapter

This chapter is an addition to the specification of the CSM module. Whereas the other parts of the specification define the behavior and the C-interfaces of the corresponding basic software module, this chapter formally specifies the corresponding AUTOSAR Service in terms of the SWC Template. The interfaces described here will be visible on the VFB and are used to generate the RTE between application software and the CSM module.

11.2 Specification of the Ports and Port Interfaces

11.2.1 General approach

It is mandatory to model the requests issued from a client to the CSM module by ports with client/server interfaces.

A client of the application domain needs the CSM module to access the different cryptographical services. To request a service via the C-API, the caller needs to give the configuration identifier as a first argument.

In order to keep the client code independent from the configured service configuration IDs, the configuration IDs are not passed from the clients to the CSM module, but are modeled as “port defined argument values” of the Provide Ports on the CSM module side. As a consequence, the configuration ID will not show up as an argument in the operations of the client-server interface. As a further consequence of this approach, there will be separate ports for each service configuration both on the client side as well as on the server side.

To receive the result of the request via a callback function, the application has to provide a Provide-Port. This Port has to be connected to the corresponding Require-Port of the CSM. In each service configuration one callback function is configured. This means, that the CSM has to provide one Require-Port for each service configuration.

The CSM specific part of the AUTOSAR configuration tool shall generate a Software-Component-Description (SWCD), which describes all elements of the AUTOSAR service implemented by the CSM. This SWCD is part of the input to the RTE generator.

There are two ways how the CSM service ports can be accessed via the RTE:

1) “direct API”:
   If the “direct API” of the RTE can be used, the call of the RTE in the client code can be optimized to “zero overhead”. The call of a service will appear as a simple macro, which will be expanded to a direct function call of the CSM

module. One condition for a direct call is the location of the client and the CSM module on the same ECU.
In this case, the CSM can be used as well in synchronous and in asynchronous mode.

2) "indirect API":
If the "indirect API" of the RTE is used, the RTE provides buffers to handle the data transfer between client and server. Indirect API calls are used, if e.g. the client and the CSM module are located on different ECUs. A buffer, that is provided by the RTE and has to be filled by the CSM, is only valid until the API of the CSM returns.
Therefore, the CSM can only be used in synchronous mode.

For more details on the RTE refer to [4].

11.2.2 Data Types

This chapter describes the data types, which will be used in the port interfaces.

The data types `uint8, uint16` and `uint32` refer to the basic AUTOSAR data type.

The data type `DataLengthPtr` refers to an address and is defined as follows:
`uint32* DataLengthPtr;`

The data type `DataPtr` refers to an address and is defined as follows:
`Uint8* DataPtr;`

The data type `Csm_ReturnType` indicates the result of a service request. This data type is defined as follows:

```
IntegerType Csm_ReturnType  {
     lowerLimit = 0
     upperLimit = 4
};
```

The following constants are associated with the data type `Csm_ReturnType` by an associated `CompuMethod`:

```
0 -> CSM_E_OK
1 -> CSM_E_NOT_OK
2 -> CSM_E_BUSY
3 -> CSM_E_SMALL_BUFFER
4 -> CSM_E_ENTROPY_EXHAUSTION
```

The data type `Csm_VerifyResultType` indicates the result of a verification operation (used by services MacVerify and SignatureVerify). This data type is defined as follows:

```
IntegerType Csm_VerifyResultType {
     lowerLimit = 0
     upperLimit = 1
};
```

The following constants are associated with the data type `Csm_VerifyResultType` by an associated `CompuMethod`:

```
     0 -> CSM_E_VER_OK
     1 -> CSM_E_VER_NOT_OK
```

The data type `Csm_ConfigIdType` identifies a configuration data set, which is unique within a service. The data type is defined as follows:

```
unit16 ConfigIdType;
```

To access the data buffers we need to define ArrayTypes for each kind of a buffer. In this definition of an array the size of the buffer (i.e. the size of the array) must be given.

```
ArrayType <BufferName>
{
     elementType = uint8;
     maxNumberOfElements = <xx>;
}

   [<xx> denotes the size of the buffer]
```

This is an example for the buffer that provides a maximum of 256 byte of input data for the hash generation.

```
ArrayType HashDataBuffer
{
     elementType = uint8;
     maxNumberOfElements = 256;
}
```

[**CSM0803**]⌈ Arrays of that kind are necessary for the following buffers:

```
     - HashDataBuffer
     - HashResultBuffer
     - HashLengthBuffer
     - MacGenerateDataBuffer
     - MacGenerateResultBuffer
     - MacGenerateLengthBuffer
     - MacVerifyDataBuffer
```

- MacVerifyCompareMacBuffer
- MacVerifyResultBuffer
- RandomSeedDataBuffer
- RandomGenerateResultBuffer
- SymBlockEncryptDataBuffer
- SymBlockEncryptResultBuffer
- SymBlockEncryptLengthBuffer
- SymBlockDecryptDataBuffer
- SymBlockDecryptResultBuffer
- SymBlockDecryptLengthBuffer
- SymEncryptInitVectorBuffer
- SymEncryptDataBuffer
- SymEncryptResultBuffer
- SymEncryptLengthBuffer
- SymDecryptInitVectorBuffer
- SymDecryptDataBuffer
- SymDecryptResultBuffer
- SymDecryptLengthBuffer
- AsymEncryptDataBuffer
- AsymEncryptResultBuffer
- AsymEncryptLengthBuffer
- AsymDecryptDataBuffer
- AsymDecryptResultBuffer
- AsymDecryptLengthBuffer
- SignatureGenerateDataBuffer
- SignatureGenerateResultBuffer
- SignatureGenerateLengthBuffer
- SignatureVerifyDataBuffer
- SignatureVerifyCompareSignatureBuffer
- SignatureVerifyResultBuffer
- ChecksumDataBuffer
- ChecksumResultBuffer
- ChecksumLengthBuffer
- KeyDerivePasswordBuffer
- KeyDeriveSaltBuffer
- KeyExchangeOwnPublicValueBuffer
- KeyExchangePartnerPublicValueBuffer
- KeyExchangeSharedSecretBuffer
- KeyExchangeSharedSecretLength
- SymKeyExtractDataBuffer
- SymKeyWrapSymDataBuffer
- SymKeyWrapAsymDataBuffer
- AsymPublicKeyExtractDataBuffer
- AsymPrivateKeyExtractDataBuffer
- AsymPrivateKeyWrapSymDataBuffer
- AsymPrivateKeyWrapAsymDataBuffer

」()

For more complex types or structures, a RecordType can be used.

```
RecordType <RecordName>
{
     RecordElement <ELEMENT_1>
     {
          elementType = <TYPE_OF_ELEMENT_1>;
     }
     RecordElement <ELEMENT_2>
     {
          elementType = <TYPE_OF_ELEMENT_2>;
     }
}
```

The elements of the RecordType have to be defined according to the type definitions in chapter 8.

This is an example for the buffer that provides an asynchronous private key with a maximum of 1024 byte.

```
RecordType AsymPrivateKeyType
{
     RecordElement length
     {
          elementType = uint32;
     }
     RecordElement AsymPrivateKeyData
     {
          elementType = ArrayType;
     }
}

ArrayType AsymPrivateKeyData
{
     elementType = uint8;
     maxNumberOfElements = 1024;
}
```

[**CSM0802**]「 Records of that kind are necessary for the following elements:

```
- SymKeyType
- AsymPublicKeyType
- AsymPrivateKeyType
- KeyExchangeBaseType
- KeyExchangePrivateType
- KeyDeriveSymKeyCustBuffer
```

」()

Restrictions:

As the RTE does not allow pointer types to be used as a parameter, we have to use arrays. The maximum size of an array has to be known at compile time. Thus it has to be configured statically

The C-API of the CSM-Services have types like “uint8*”, i.e. a pointer to the first element of an array. The RTE can only handle a pointer to an array-type, but not a pointer to an element of an array. Thus the CSM module has to provide a wrapper, that maps the pointer to an array to a pointer of an element of an array.

As the RTE only knows the array type, the application has to provide a buffer which is exactly as big as the array. The buffer must not be smaller.

These restrictions are also valid for record types.

11.2.3 Port Interfaces

For each service configurations, a separate port is generated. All operations related to a service configuration can be accessed via this port. This Port is assigned to a Client/Server-Interface, which contains all service operations related to the corresponding service.
.

```
ClientServerInterface Csm<Service> {
      PossibleErrors {
            <all possible Return values Csm_ReturnType of the corresponding
            APIs      Csm_<Service>Start,      Csm_<Service>Update      and
            Csm_<Service>Finish>
      };

     <Service>Start ( < all parameters of the API
                        Csm_<Service>Start, except cfgId > );
     <Service>Update( < all parameters of the API
                        Csm_<Service>Update > );
     <Service>Finish( < all parameters of the API
                        Csm_<Service>Finish > );
     <Service>      ( < all parameters of the API
                        Csm_<Service> > );
};
```

11.2.3.1 Hash Interface

[**CSM0775**] ⌈

```
ClientServerInterface CsmHash {
    PossibleErrors {
        CSM_E_NOT_OK       = 1
        CSM_E_BUSY         = 2
        CSM_E_SMALL_BUFFER = 3
    };

    HashStart (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
```

```
    );

    HashUpdate (
        IN    HashDataBuffer    dataBuffer,
        IN    uint32            dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    HashFinish (
        OUT   HashResultBuffer  resultBuffer,
        INOUT HashLengthBuffer  resultLength,
        IN    boolean           TruncationIsAllowed,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};
```

11.2.3.2 MacGenerate Interface

[**CSM0776**]

```
ClientServerInterface CsmMacGenerate {
    PossibleErrors {
        CSM_E_NOT_OK       = 1
        CSM_E_BUSY         = 2
        CSM_E_SMALL_BUFFER = 3
    };

    MacGenerateStart (
        IN    SymKeyType        key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    MacGenerateUpdate (
        IN    MacGenerateDataBuffer  dataBuffer,
        IN    uint32                 dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    MacGenerateFinish (
        OUT   MacGenerateResultBuffer   resultBuffer,
        INOUT MacGenerateLengthBuffer   resultLength,
        IN    boolean                   TruncationIsAllowed,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};
```

11.2.3.3 MacVerify Interface

[**CSM0777**]「

```
ClientServerInterface CsmMacVerify {
    PossibleErrors {
        CSM_E_NOT_OK    = 1
        CSM_E_BUSY      = 2
    };

    MacVerifyStart (
        IN    SymKeyType        key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    MacVerifyUpdate (
        IN    MacVerifyDataBuffer    dataBuffer,
        IN    uint32                 dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    MacVerifyFinish (
        IN    MacVerifyCompareBuffer    MacBuffer,
        IN    uint32                    MacLength,
        OUT   MacVerifyResultBuffer     resultBuffer,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};」()
```

11.2.3.4 RandomSeed Interface

[CSM0778]「

```
ClientServerInterface CsmRandomSeed {
    PossibleErrors {
        CSM_E_NOT_OK    = 1
        CSM_E_BUSY      = 2
    };

    RandomSeedStart (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    RandomSeedUpdate (
        IN    RandomSeedDataBuffer  seedBuffer,
        IN    uint32                seedLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    RandomSeedFinish (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
```

```
    );
};」()
```

11.2.3.5 RandomGenerate Interface

```
[CSM0779]「
ClientServerInterface CsmRandomGenerate {
    PossibleErrors {
        CSM_E_NOT_OK             = 1
        CSM_E_BUSY               = 2
        CSM_E_ENTROPY_EXHAUSTION = 4
    };

    RandomGenerate (
        OUT   RandomGenerateResultBuffer  resultBuffer,
        IN    uint32                      resultLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_ENTROPY_EXHAUSTION)
    };
};」()
```

11.2.3.6 SymBlockEncrypt Interface

```
[CSM0780]「
ClientServerInterface CsmSymBlockEncrypt {
    PossibleErrors {
        CSM_E_NOT_OK           = 1
        CSM_E_BUSY             = 2
        CSM_E_SMALL_BUFFER     = 3
    };

    SymBlockEncryptStart (
        IN    SymKeyType           key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymBlockEncryptUpdate (
        IN    SymBlockEncryptDataBuffer   plainTextBuffer,
        IN    uint32                      plainTextLength,
        OUT   SymBlockEncryptResultBuffer cipherTextBuffer,
        INOUT SymBlockEncryptLengthBuffer cipherTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );

    SymBlockEncryptFinish (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};」()
```

11.2.3.7 SymBlockDecrypt Interface

[**CSM0781**]⌈

```
ClientServerInterface CsmSymBlockDecrypt {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    SymBlockDecryptStart (
        IN    SymKeyType           key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymBlockDecryptUpdate (
        IN    SymBlockDecryptDataBuffer   cipherTextBuffer,
        IN    uint32                      cipherTextLength,
        OUT   SymBlockDecryptResultBuffer plainTextBuffer,
        INOUT SymBlockDecryptLengthBuffer plainTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );

    SymBlockDecryptFinish (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌋()
```

11.2.3.8 SymEncrypt Interface

[**CSM0782**]⌈

```
ClientServerInterface CsmSymEncrypt {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    SymEncryptStart (
        IN    SymKeyType                  key,
        IN    SymEncryptInitVectorBuffer  InitVectorBuffer,
        IN    uint32                      InitVectorLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymEncryptUpdate (
        IN    SymEncryptDataBuffer     plainTextBuffer,
        IN    uint32                   plainTextLength,
        OUT   SymEncryptResultBuffer   cipherTextBuffer,
```

```
        INOUT SymEncryptLengthBuffer    cipherTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );

    SymEncryptFinish (
        OUT   SymEncryptResultBuffer    cipherTextBuffer,
        INOUT SymEncryptLengthBuffer    cipherTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};⌟()
```

11.2.3.9 SymDecrypt Interface

[**CSM0783**]⌜

```
ClientServerInterface CsmSymDecrypt {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    SymDecryptStart (
        IN    SymKeyType                    key,
        IN    SymDecryptInitVectorBuffer    InitVectorBuffer,
        IN    uint32                        InitVectorLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymDecryptUpdate (
        IN    SymDecryptDataBuffer          cipherTextBuffer,
        IN    uint32                        cipherTextLength,
        OUT   SymDecryptResultBuffer        plainTextBuffer,
        INOUT SymDecryptLengthBuffer        plainTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );

    SymDecryptFinish (
        OUT   SymDecryptResultBuffer        plainTextBuffer,
        INOUT SymDecryptLengthBuffer        plainTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};⌟()
```

11.2.3.10 AsymEncrypt Interface

[**CSM0784**]⌜

```
ClientServerInterface CsmAsymEncrypt {
    PossibleErrors {
```

```
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    AsymEncryptStart (
        IN    AsymPublicKeyType     key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymEncryptUpdate (
        IN    AsymEncryptDataBuffer    plainTextBuffer,
        IN    uint32                   plainTextLength,
        OUT   AsymEncryptResultBuffer  cipherTextBuffer,
        INOUT AsymEncryptLengthBuffer  cipherTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );

    AsymEncryptFinish (
        OUT   AsymEncryptResultBuffer  cipherTextBuffer,
        INOUT AsymEncryptLengthBuffer  cipherTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};⌟()
```

11.2.3.11 AsymDecrypt Interface

[**CSM0785**]⌜

```
ClientServerInterface CsmAsymDecrypt {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    AsymDecryptStart (
        IN    AsymPrivateKeyType   key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymDecryptUpdate (
        IN    AsymDecryptDataBuffer    cipherTextBuffer,
        IN    uint32                   cipherTextLength,
        OUT   AsymDecryptResultBuffer  plainTextBuffer,
        INOUT AsymDecryptLengthBuffer  plainTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );

    AsymDecryptFinish (
```

```
        OUT   AsymDecryptResultBuffer  plainTextBuffer,
        INOUT AsymDecryptLengthBuffer  plainTextLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    };
); ⌟()
```

11.2.3.12 SignatureGenerate Interface

[**CSM0786**]⌜

```
ClientServerInterface CsmSignatureGenerate {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    SignatureGenerateStart (
        IN    AsymPrivateKeyType   key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SignatureGenerateUpdate (
        IN    SignatureGenerateDataBuffer  dataBuffer,
        IN    uint32                       dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SignatureGenerateFinish (
        OUT   SignatureGenerateResultBuffer  resultBuffer,
        INOUT SignatureGenerateLengthBuffer  resultLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};⌟()
```

11.2.3.13 SignatureVerify Interface

[**CSM0787**]⌜

```
ClientServerInterface CsmSignatureVerify {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    SignatureVerifyStart (
        IN    AsymPublicKeyType    key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
```

```
    SignatureVerifyUpdate (
        IN   SignatureVerifyDataBuffer    dataBuffer,
        IN   uint32                       dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SignatureVerifyFinish (
        IN    SignatureVerifyCompareSignatureBuffer signatureBuffer,
        IN    uint32                                signatureLength,
        OUT   SignatureVerifyResultBuffer           resultBuffer,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.14 Checksum Interface

[**CSM0788**]⌜

```
ClientServerInterface CsmChecksum {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    ChecksumStart (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    ChecksumUpdate (
        IN   ChecksumDataBuffer    dataBuffer,
        IN   uint32                dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    ChecksumFinish (
        OUT   ChecksumResultBuffer        resultBuffer,
        INOUT ChecksumLengthBuffer        resultLength,
        IN    boolean                     TruncationIsAllowed,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};⌟()
```

11.2.3.15 KeyDerive Interface

[**CSM0789**]⌜

```
ClientServerInterface CsmKeyDerive {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
```

```
        CSM_E_BUSY              = 2
    };

    KeyDeriveStart (
        IN    uint32              keyLength,
        IN    uint32              iterations,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    KeyDeriveUpdate (
        IN    KeyDerivePasswordBuffer  passwordBuffer,
        IN    uint32                   passwordLength,
        IN    KeyDeriveSaltBuffer      saltBuffer,
        IN    uint32                   saltLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    KeyDeriveFinish (
        INOUT SymKeyType         key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.16 KeyDeriveSymKey Interface

[**CSM0790**]⌜

```
ClientServerInterface CsmKeyDeriveSymKey {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    KeyDeriveSymKey (
        IN    SymKeyType                baseKey,
        IN    KeyDeriveSymKeyCustBuffer customisationValBuffer,
        IN    uint32                    customisationValLength,
        INOUT SymKeyType                derivedKey,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.17 KeyExchangeCalcPubVal Interface

[**CSM0791**]⌜

```
ClientServerInterface CsmKeyExchangeCalcPubVal {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
```

```
        CSM_E_SMALL_BUFFER      = 3
    };

    KeyExchangeCalcPubVal
    (
        IN    KeyExchangeBaseType                baseBuffer,
        IN    KeyExchangePrivateType             privateValueBuffer,
        OUT   KeyExchangeOwnPublicValueBuffer    publicValueBuffer,
        INOUT KeyExchangeOwnPublicValueSizeBuffer publicValueLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};⌟()
```

11.2.3.18 KeyExchangeCalcSecret Interface

[**CSM0792**]⌜

```
ClientServerInterface CsmKeyExchangeCalcSecret {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
        CSM_E_SMALL_BUFFER      = 3
    };

    KeyExchangeCalcSecretStart (
        IN    KeyExchangeBaseType      baseBuffer,
        IN    KeyExchangePrivateType   privateValueBuffer,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    KeyExchangeCalcSecretUpdate (
        IN  KeyExchangePartnerPublicValueBuffer partnerPublicValueBuffer,
        IN  uint32                              partnerPublicValueLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    KeyExchangeCalcSecretFinish (
        OUT   KeyExchangeSharedSecretBuffer   sharedSecretBuffer,
        INOUT KeyExchangeSharedSecretLength   sharedSecretLength,
        IN    boolean                         TruncationIsAllowed,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY, CSM_E_SMALL_BUFFER)
    );
};⌟()
```

11.2.3.19 KeyExchangeCalcSymKey Interface

[**CSM0793**]⌜

```
ClientServerInterface CsmKeyExchangeCalcSymKey {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
```

```
        CSM_E_BUSY              = 2
    };

    KeyExchangeCalcSymKeyStart (
        IN    KeyExchangeBaseType      baseBuffer,
        IN    KeyExchangePrivateType   privateValueBuffer,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    KeyExchangeCalcSymKeyUpdate (
        IN  KeyExchangePartnerPublicValueBuffer partnerPublicValueBuffer,
        IN  uint32                              partnerPublicValueLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    KeyExchangeCalcSymKeyFinish (
        INOUT SymKeyType               sharedSecretLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.20 SymKeyExtract Interface

[**CSM0794**]⌜

```
ClientServerInterface CsmSymKeyExtract {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    SymKeyExtractStart
    (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymKeyExtractUpdate (
        IN    SymKeyExtractDataBuffer  dataBuffer,
        IN    uint32                   dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymKeyExtractFinish (
        INOUT SymKeyType               key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.21 SymKeyWrapSym Interface

[**CSM0795**]⌈

```
ClientServerInterface CsmSymKeyWrapSym {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    SymKeyWrapSymStart
    (
        IN    SymKeyType              key,
        IN    SymKeyType              wrappingKey,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymKeyWrapSymUpdate (
        OUT   SymKeyWrapSymDataBuffer  dataBuffer,
        INOUT DataLengthPtr            dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymKeyWrapSymFinish (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌋()
```

11.2.3.22 SymKeyWrapAsym Interface

[**CSM0796**]⌈

```
ClientServerInterface CsmSymKeyWrapAsym {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    SymKeyWrapAsymStart
    (
        IN    SymKeyType              key,
        IN    AsymPublicSymKeyType    wrappingKey,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    SymKeyWrapAsymUpdate (
        OUT   SymKeyWrapAsymDataBuffer dataBuffer,
        INOUT DataLengthPtr            dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
```

```
    SymKeyWrapAsymFinish (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.23 AsymPublicKeyExtract Interface

[**CSM0797**]⌈

```
ClientServerInterface CsmAsymPublicKeyExtract {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    AsymPublicKeyExtractStart
    (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymPublicKeyExtractUpdate (
        IN    AsymPublicKeyExtractDataBuffer dataBuffer,
        IN    uint32                         dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymPublicKeyExtractFinish (
        INOUT AsymPublicKeyType         key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.24 AsymPrivateKeyExtract Interface

[**CSM0798**]⌈

```
ClientServerInterface CsmAsymPrivateKeyExtract {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    AsymPrivateKeyExtractStart (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymPrivateKeyExtractUpdate (
        IN    AsymPrivateKeyExtractDataBuffer   dataBuffer,
        IN    uint32                            dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
```

```
    );

    AsymPrivateKeyExtractFinish (
        INOUT AsymPrivateKeyType        key,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.25 AsymPrivateKeyWrapSym Interface

[**CSM0799**]⌈

```
ClientServerInterface CsmAsymPrivateKeyWrapSym {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    AsymPrivateKeyWrapSymStart
    (
        IN    AsymPrivateKeyType       key,
        IN    SymKeyType               wrappingKey,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymPrivateKeyWrapSymUpdate (
        OUT   AsymPrivateKeyWrapSymDataBuffer   dataBuffer,
        INOUT DataLengthPtr                     dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymPrivateKeyWrapSymFinish (
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.26 AsymPrivateKeyWrapAsym Interface

[**CSM0800**]⌈

```
ClientServerInterface CsmAsymPrivateKeyWrapAsym {
    PossibleErrors {
        CSM_E_NOT_OK            = 1
        CSM_E_BUSY              = 2
    };

    AsymPrivateKeyWrapAsymStart
    (
        IN    AsymPrivateKeyType       key,
        IN    AsymPublicKeyType        wrappingKey,
```

```
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymPrivateKeyWrapAsymUpdate (
        OUT   AsymPrivateKeyWrapAsymDataBuffer  dataBuffer,
        INOUT DataLengthPtr                     dataLength,
        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );

    AsymPrivateKeyWrapAsymFinish (

        ERR(CSM_E_NOT_OK, CSM_E_BUSY)
    );
};⌟()
```

11.2.3.27 Callback Interface

To access the configured callback functions, the CSM provides one Port per configuration. This Port is assigned to a Client/Server-Interface, which contains

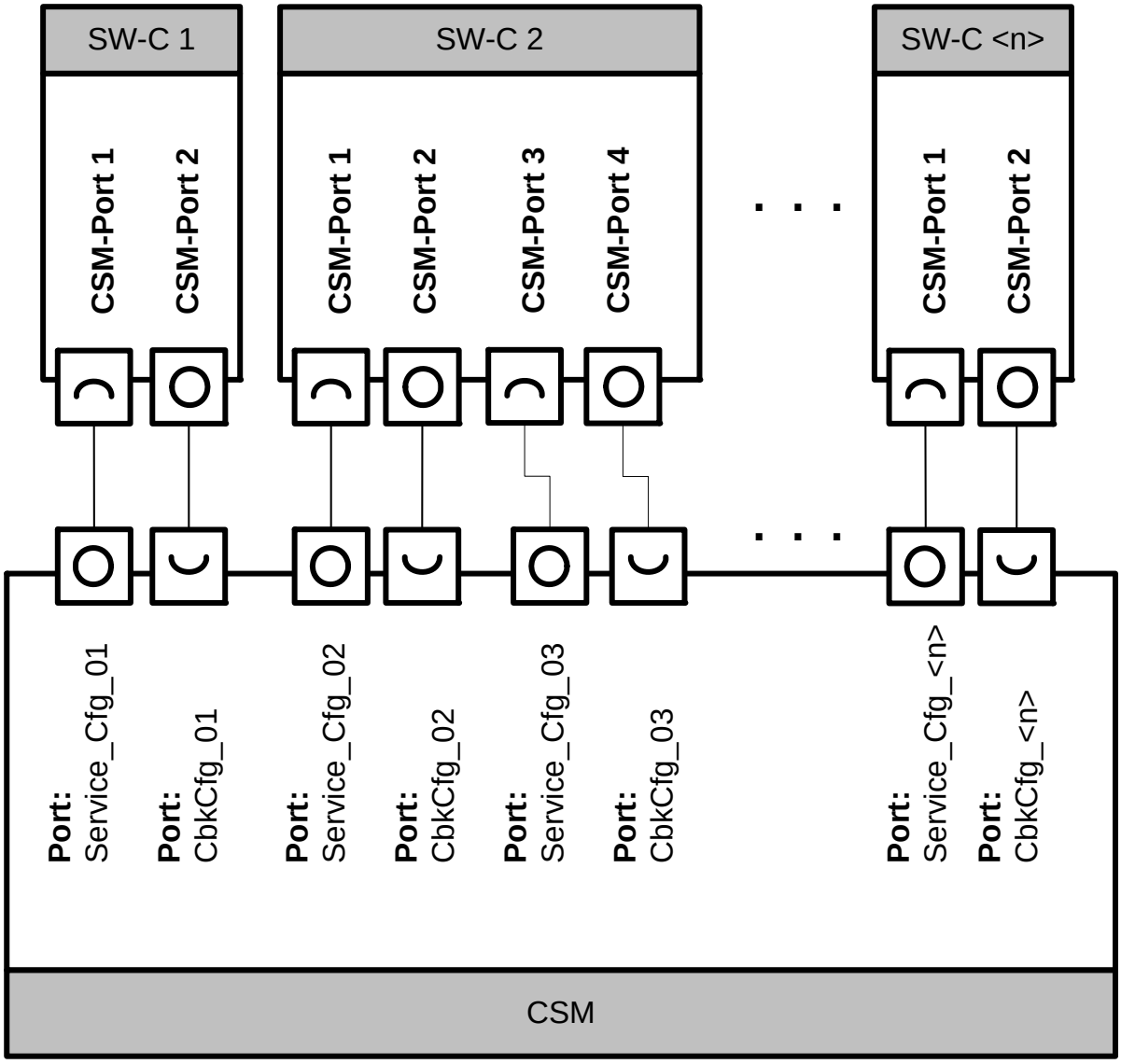
[**CSM0801**]⌜

```
ClientServerInterfaces CsmCallback {
    PossibleErrors {
        CSM_E_NOT_OK     = 1
    };

    JobFinished (
        IN Csm_ReturnType   retVal,
        ERR(CSM_E_NOT_OK)
    );
};⌟()
```

11.2.4 Ports

The following figure shows how AUTOSAR Software Components are connected to the CSM module via Client/Server-Ports.



On the CSM module side, one separate Port for each service configuration is provided. As each configuration provides a separate callback function, there has to be also one separate Port for each service configuration to access the callback functions of the applications.

The name of a service configuration is configured by the user, whereas the numerical configuration ID is generated by the Csm module. The names of the ports shall be derived from the name of the corresponding service configuration. Thus it makes it easier for the user to identify the correct port, to which the SWC has to be connected. In addition, the name of the configuration does not change, when another configuration is added or removed, whereas the configuration ID might change.

Example:

```
P-Port-Prototype DemoApplication_HashCfg {

    /* operation prototypes */
```

```
    HashStart;
    HashUpdate;
    HashFinish;

    /* related interface */
    CsmHash
};
```

Restriction:
An Application cannot use a configuration that is already used by another application.

11.3 Internal Behaviour

```
InternalBehavior CsmIntBeh {
    /* definition of associated operation-invoked RTE-events
    not shown (it is done in the same way as for any SWC type)
    */

    /* section “runnable entities”: */
    RunnableEntity RE_<Service>Start
            Symbol “Csm_Rte<Service>Start”
            canbeInvokedConcurrently = false

    RunnableEntity RE_<Service>Update
            Symbol “Csm_Rte<Service>Update”
            canbeInvokedConcurrently = false

    RunnableEntity RE_<Service>Finish
            Symbol “Csm_Rte<Service>Finish”
            canbeInvokedConcurrently = false

    RunnableEntity RE_<Service>
            Symbol “Csm_Rte<Service>”
            canbeInvokedConcurrently = false


    // port defined argument for each port
    PortArgument { port = <PortName>
                   Value.type = Csm_ConfigIdType
                   Value.value = <configID> )

    /* end of section “runnable entities” */

    );
```

Restriction:

The RTE pastes the PortDefinedArgument as a first parameter to all operations of the Port. As the PortDefinedArgument for the configuration ID is not needed by all operations, the CSM module has to provide a wrapper functionality, which filters out the PortDefinedArgument for operations that do not need it.

Examples:

```
Std_ReturnType Csm_RteHashStart( ConfigIdType pdav )
{
      Csm_HashStart( pdav );
}

Std_ReturnType Csm_RteHashUpdate( ConfigIdType pdav,
                                  HashDataBuffer dataPtr,
                                  uint32 dataLength )
{
      Csm_HashUpdate( (uint8*) dataPtr, dataLength );
}
```

Note that the wrapper functionality is also necessary for mapping the pointer to arrays to pointer to elements of arrays (see 11.2.2)

11.4 Configuration of the Configuration IDs

The configuration IDs of the CSM module are modeled as “port defined argument values”: The CSM specific part of the AUTOSAR configuration tool shall generate a Software-Component-Description file. The argument values, i.e. the configuration IDs, have to be derived from the service configurations, that have been configured by the user. Note that the ports visible on the client side are not affected by this.

Example:

The user set up a configuration for the hash service. The name of the configuration was set to “DemoApplication_HashCfg”.

The CSM internally assigns a numeric value to this name:
`#define DemoApplication_HashCfg_Id  1`

In the generated Software-Component-Description file the corresponding port shall be named:
`DemoApplication_HashCfg`

The corresponding port defined argument value shall be set to: `1`