MICROSAR Classic CSM

Technical Reference

Cryptographic Service Manager

Version 7.02.01

Document Information

History

Author	Date	Version	Remarks
vismss	2017-03-14	1.01.00	Initial Creation for ASR 4.3
visgut	2017-03-17	1.01.00	Add configurational details
vismss	2017-05-08	1.02.00	2.1.1 Added backward compatible static definition limitation 4 Adapted to Specification
vismss	2017-06-07	1.03.00	Adapted chapter 3.2 Added information about SecureCounter Removed API description for SecureCounter
vismaw	2018-08-27	3.01.00	Added information about Pre-and PostJob Callouts Added API description for Csm_MainFunction
vismwe	2018-10-23	3.01.01	Change E_PENDING to CSM_E_PENDING
vismaw	2019-03-12	4.01.00	Added Csm_KeyElementCopyPartial Added Redirection
visenc, visewl, vismxe	2019-11-11	5.00.00	Added Csm_JobKeySetValid Added Csm_JobRandomSeed Added Csm_JobKeyGenerate Added Csm_JobKeyDerive Added Csm_JobKeyExchangeCalcPubVal Added Csm_JobKeyExchangeCalcSecret. Added chapter 4.5.
visewl	2020-03-12	5.00.02	Added Table 2-8.
visrpp	2020-07-08	5.01.00	Changed chapter 4.2.5
vismxe	2021-01-15	6.00.00	Removed obsolete chapters “Backward Compatibility” and “Configuration with DaVinci Configurator 5 Pro”
visenc	2021-03-22	6.01.00	Changed chapter 2.1, 2.4.1 and 4.1 Added chapter 4.2.38 and 4.2.39
visewl	2021-12-15	6.02.00	Add caution boxes in 4.4.1 and in 4.5 Removed Chapter “Component History”
visewl visrpp	2023-03-13	7.00.00	Product name updated to MICROSAR Classic Added chapter 2.1.2, 2.4.2, 2.6, 4.2.8, 4.2.9, 4.2.32, 4.5.4 Update chapter 2.1.1, 3.1, 4.2
visfnn	2023-04-13	7.01.00	Add multicore / partitioning support (chapter 2.7)

mapelt	2023-08-03	7.02.00	Added chapter 6 Add Csm_InitMemory description in Chapter 2.7.1
Mapelt, vismwe	2024-02-27	7.02.01	Revised chapter 2.4.2 and 6, Added chapter 2.8, Revised chapter 4.4 and 4.5 for chapter 2.8 Sync chapter 4 with code

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_CryptoServiceManager.pdf	4.3.0
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.3.0
[3]	AUTOSAR	AUTOSAR_SWS_CryptoServiceManager.pdf	4.4.0
[4]	AUTOSAR	AUTOSAR_SWS_CryptoServiceManager.pdf	R19-11
[5]	AUTOSAR	AUTOSAR_SWS_CryptoServiceManager.pdf	R20-11
[6]	AUTOSAR	AUTOSAR_SWS_CryptoServiceManager.pdf	R21-11

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector’s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CSM as specified in [1], [3], [4], [5] and [6].

Supported Configuration Variants:	pre-compile	
Vendor ID:	CSM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CSM_MODULE_ID	110 decimal

The CSM provides synchronous and asynchronous services to enable a unique access to basic cryptographic functionalities for software components (SWC) and basic software (BSW). The CSM offers a standardized interface to higher software layers to access these functionalities.

1.1 Architecture Overview

The following figure shows where the CSM is located in the AUTOSAR architecture.

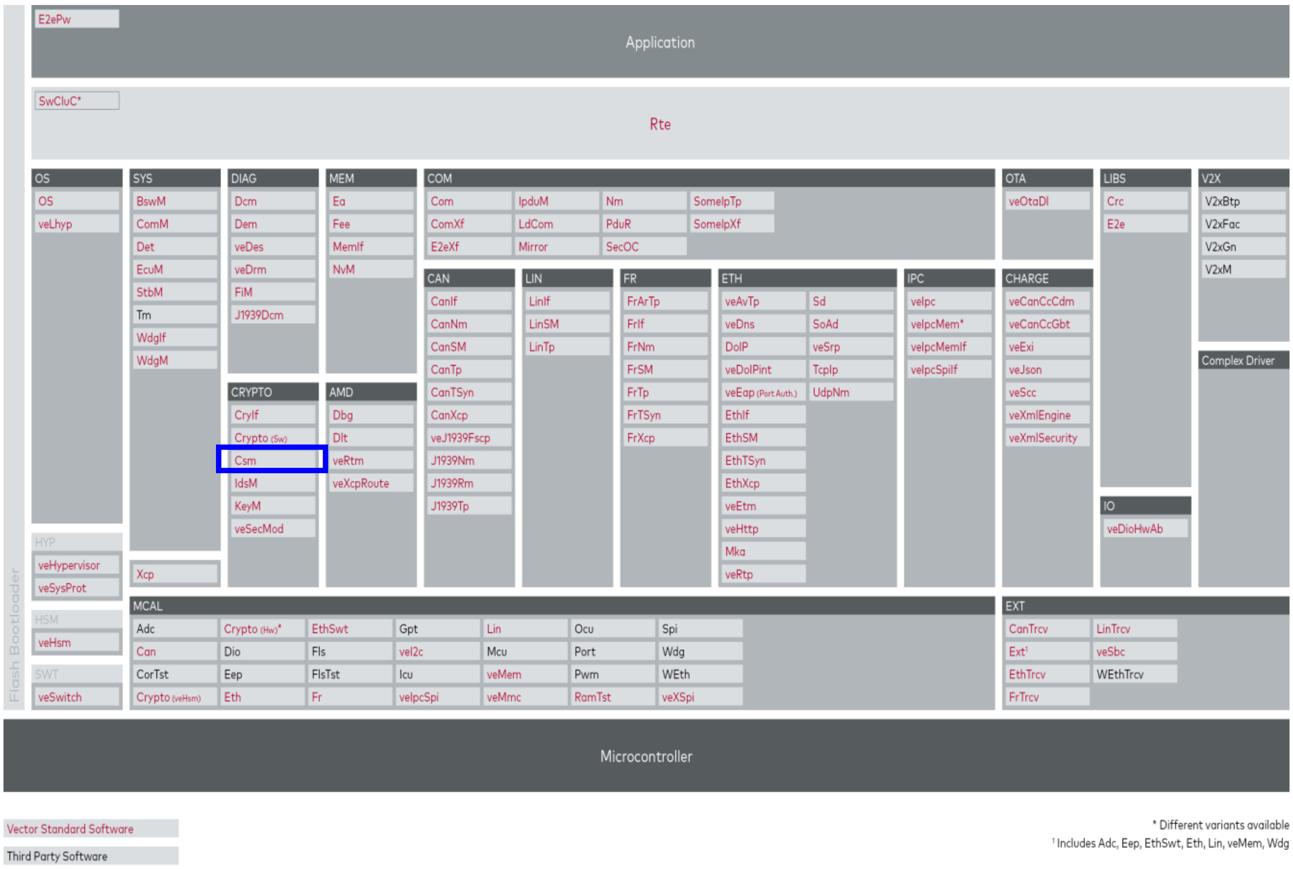


Figure 1-1 AUTOSAR 4.3 Architecture Overview

The next figure shows the interfaces to adjacent modules of the CSM. These interfaces are described in chapter 4.

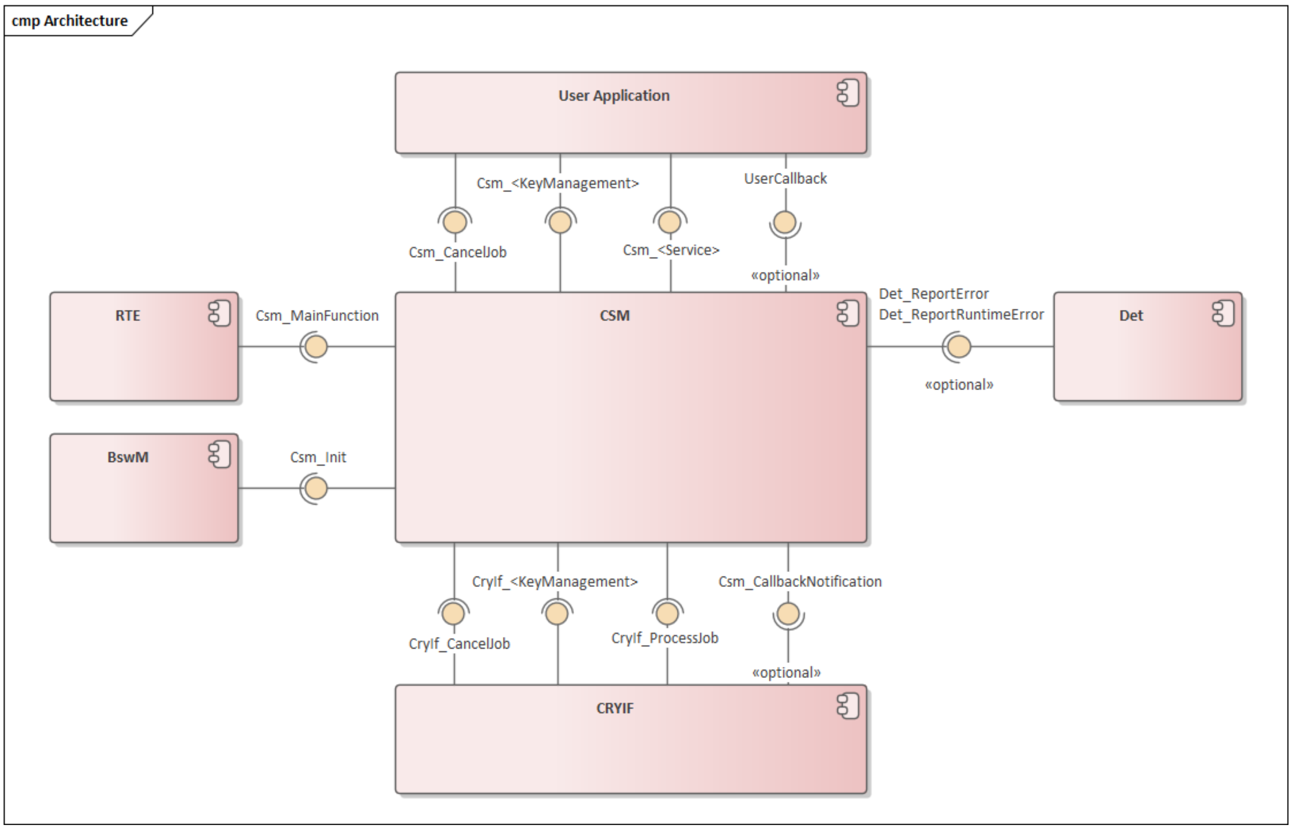


Figure 1-2 Interfaces to adjacent modules of the CSM

2 Functional Description

2.1 Features

The features listed in the following tables cover the complete functionality specified for the CSM.

The AUTOSAR standard functionality is specified in [1], [3], [4], [5] and [6], the corresponding features are listed in the following tables.

The following features specified in [1] are supported:

Supported AUTOSAR 4.3.x Standard Conform Features
Queueing and cancellation of jobs
Job prioritization
Synchronous and asynchronous job handling
Key management APIs

Table 2-1 Supported AUTOSAR standard conform features according to 4.3.x

The following features specified in [3] are supported:

Supported AUTOSAR 4.4.x Standard Conform Features
Redirection of key elements
JobKey primitives for Asynchronous Key Handling

Table 2-2 Supported AUTOSAR standard conform features according to 4.4.x

The following features specified in [5] are supported:

Supported AUTOSAR R20-11 Standard Conform Features
Save and restore context for running crypto operations
KeyGetStatus and KeySetInvalid services

Table 2-3 Supported AUTOSAR standard conform features according to R20-11

2.1.1 Deviations

The features listed in the following tables describe the not supported features or deviations.

The following features specified in [1] are not supported:

Not Supported AUTOSAR 4.3.x Standard Conform Features
4.x backward compatible API and Client-Server Interface
4.x backward compatible definitions are static and will be removed in the next release
Secure Counter Increment and Secure Counter Read are not supported

Table 2-4 Not supported AUTOSAR standard conform features according to 4.3.x

The following features specified in [3] are not supported:

Not Supported AUTOSAR 4.4.x Standard Conform Features
The primitives Csm_JobCertificateParse and Csm_JobCertificateVerify are not supported

Table 2-5 Not supported AUTOSAR standard conform features according to 4.4.x

There are the following deviations since [1]:

Deviations from AUTOSAR Standard Conform Features
The values of CRYPTO_ALGOMODE_XTS and CRYPTO_ALGOMODE_GCM are interchanged

Table 2-6 Deviations from all AUTOSAR Standard Conform Features since 4.3.x

There are the following deviations from [1]:

Deviations from AUTOSAR 4.3.x Standard Conform Features
Usage of CRYPTO_E_BUSY instead of CRYPTO_E_QUEUE_FULL
The CallbackNotification service interface does not have the parameter job of type Crypto_JobType

Table 2-7 Deviations from AUTOSAR Standard Conform Features according to 4.3.x

There are the following deviations from [3]:

Deviations from AUTOSAR 4.4.x Standard Conform Features
Using additionally introduced types Csm_const_DataPtr, Csm_LengthPtr and Csm_VerifyResultPtr and setting all parameters' directions to IN for operations of the Client-Server-Interfaces (DATA_REFERENCES) and the Client-Server-Interfaces (Key Management) of the service interface
Using Csm_KeyDataType_{Crypto} instead of Csm_DataPtr for the operations KeyExchangeCalcPubVal, KeyExchangeCalcSecret and RandomSeed of the Client-Server-Interfaces (Key Management) of the service interface

Deviations from AUTOSAR 4.4.x Standard Conform Features
Usage of CRYPTO_E_BUSY instead of CRYPTO_E_QUEUE_FULL
The Csm_Init function is provided without a pointer
The Key APIs do not verify the initialization state since they do not access any initialization related artefacts.

Table 2-8 Deviations from AUTOSAR Standard Conform Features according to 4.4.x

There are the following deviations from [4]:

Deviations from AUTOSAR R19-11 Standard Conform Features
Using additionally introduced types Csm_const_DataPtr, Csm_LengthPtr and Csm_VerifyResultPtr and setting all parameters' directions to IN for operations of the Client-Server-Interfaces (DATA_REFERENCES) and the Client-Server-Interfaces (Key Management) of the service interface
The Csm_Init function is provided without a pointer

Table 2-9 Deviations from AUTOSAR Standard Conform Features according to R19-11

There are the following deviations from [5]:

Deviations from AUTOSAR R20-11 Standard Conform Features
CSM allows a call to Csm_RestoreContextJob() if the related job is in IDLE which is a deviation from SWS_Csm_91069.
The Csm_Init function is provided without a pointer

Table 2-10 Deviations from AUTOSAR Standard Conform Features according to R20-11

There are the following deviations from [6]:

Deviations from AUTOSAR R21-11 Standard Conform Features
CSM allows a call to Csm_RestoreContextJob() if the related job is in IDLE which is a deviation from SWS_Csm_91069.
The Csm_Init function is provided without a pointer

Table 2-11 Deviations from AUTOSAR Standard Conform Features according to R21-11

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Immediate job triggering in Csm_CallbackNotification
Callback Notification also for the start of a job
Pre- and post-job callouts

Table 2-12 Features Provided Beyond the AUTOSAR Standard

2.1.2.1 Immediate job triggering in Csm_CallbackNotification

This implementation of the CSM provides an optional feature to minimize the time between the triggering of asynchronous jobs. If enabled, queued jobs will be triggered (passed to the CryIf for processing) directly in the Csm_CallbackNotification instead of delaying this to the next Csm_MainFunction call.

This behavior can be configured per queue in the configuration and is advisable for hardware based Crypto if the driver is able to already accept a new job when invoking the callback notification.

2.1.2.2 Callback Notification also for the start of a job

This implementation also allows the notification of the start of a job via callback. This applies if only the start was requested (streaming mode) and feature is enabled.

2.1.2.3 Pre- and post-job callouts

Refer to chapter 2.5.

2.2 Initialization

Before any other functionality of the CSM module can be used the initialization function `Csm_Init()` has to be called by the BSWM.

For manual null initialization of RAM variables the CSM offers the function `Csm_InitMemory()` which can be called before the `Csm_Init()`.

2.3 Main Functions

The CSM module implementation provides one main function for each configured partition. When the usage of asynchronous job processing is enabled, these main functions have to be called cyclically on task level. The main functions are responsible to dispatch new jobs to the underlying CryIf respectively Crypto Driver.

2.4 Error Handling

2.4.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `CSM_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported CSM ID is 110.

The reported service IDs identify the services which are described in 4.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Csm_Init
0x01	Csm_MainFunction
0x3B	Csm_GetVersionInfo

Service ID	Service
0x5D	Csm_Hash
0x60	Csm_MacGenerate
0x61	Csm_MacVerify
0x5E	Csm_Encrypt
0x5F	Csm_Decrypt
0x62	Csm_AEADEncrypt
0x63	Csm_AEADDecrypt
0x76	Csm_SignatureGenerate
0x64	Csm_SignatureVerify
0x65	Csm_SecureCounterIncrement
0x66	Csm_SecureCounterRead
0x72	Csm_RandomGenerate
0x78	Csm_KeyElementSet
0x67	Csm_KeySetValid
0x68	Csm_KeyElementGet
0x71	Csm_KeyElementCopy
0x73	Csm_KeyCopy
0x69	Csm_RandomSeed
0x6A	Csm_KeyGenerate
0x6B	Csm_KeyDerive
0x6C	Csm_KeyExchangeCalcPubVal
0x6D	Csm_KeyExchangeCalcSecret
0x6E	Csm_CertificateParse
0x74	Csm_CertificateVerify
0x6F	Csm_CancelJob
0x70	Csm_CallbackNotification
0x7A	Csm_JobKeySetValid
0x7B	Csm_JobRandomSeed
0x7C	Csm_JobKeyGenerate
0x7D	Csm_JobKeyDerive
0x7E	Csm_JobKeyExchangeCalcPubVal
0x7F	Csm_JobKeyExchangeCalcSecret
0x83	Csm_KeyGetStatus
0x84	Csm_JobKeySetInalid
0x85	Csm_KeySetInvalid
0x86	Csm_SaveContextJob
0x87	Csm_RestoreContextJob
0xFF	Error not related to a special service but e.g. general job processing

Table 2-13 Service IDs

The errors reported to DET are described in the following table:

Error Code	Description
0x01	API request called with invalid parameter (Nullpointer)
0x03	API request called with invalid parameter (invalid method for selected service)
0x04	API request called with Csm configuration ID out of range
0x05	API request called before initialization of CSM module
0x07	Initialization of CSM module failed
0x08	API request called with invalid processing mode
0x09	Mismatch between the called API request and the service type of the job (e.g. Csm_Hash was called with a jobId configured to an encryption primitive)
0x11	The service Csm_Init() is called while the module is already initialized
0x12	The configured job partition doesn't match the currently executed partition.

Table 2-14 Errors reported to DET

2.4.2 Runtime Error Reporting

By default, runtime errors are reported to the DET using the service `Det_ReportRuntimeError()` as specified in [2], if development error reporting is enabled.

The runtime errors reported to DET are described in the following table:

Error Code	Description
0x01	Queue overrun - More jobs have been requested than the queue can store
0x81	Unexpected callback from lower layer.

Table 2-15 Runtime errors reported to DET

2.5 Pre-and Post-Job Callouts

This feature can be enabled or disabled individually for each job. If enabled, a prejob callout is called before the corresponding job is processed. The signature for the prejob callout looks like that:

```
Std_ReturnType Csm_PreJobCalloutFunc)(
Crypto_JobType *job,
Csm_JobCalloutStateType state);
```

The first parameter is a pointer to the corresponding job which shall be processed, the second parameter is only used for asynchronous jobs. If the job is asynchronous, the behavior of the prejob callout might also be asynchronous. Therefore, it is possible to signalize the Csm that the processing of the prejob callout has not finished yet by returning CSM_E_PENDING. If CSM_E_PENDING is returned, the Csm delays the processing of the job and calls the prejob callout again until the return value is different. The parameter state indicates whether the prejob was called for the first time (INITIAL) or not (PENDING). Furthermore, the application implementing the prejob callout has the possibility to suppress processing of the job by returning E_NOT_OK. Then, only the postjob callout will be called.

The postjob callout will be called after the processing of a job. Its signature looks like that:

```
Std_ReturnType  Csm_PostJobCalloutFunc)(
Crypto_JobType *job,
Csm_JobCalloutStateType state,
Std_ReturnType *jobReturnValue);
```

The first parameter is once again the pointer to the corresponding job. The second pointer is the state in which the postjob callout is currently in. Up to now, the postjob callout does not have the possibility to be called more than once, so the state will always have the same value. Last, a pointer to the return value of the job processing is given. The postjob callout is able to manipulate the return value to an arbitrary value, which than is given to the application which originally triggered the job. Like the state parameter, the return value is given for future use, up to now the Csm will ignore it.

Caution
Both, the pre- and postjob callouts have a pointer to the corresponding job. It is possible to manipulate the job object (e.g. adapt buffers, adjust lengths, and so on). Beware that the Csm is not able to check later on whether the buffers are set up accordingly.

2.6 Support of different AUTOSAR versions

This implemenation of the Csm aims for supporting different AUTOSAR versions. Whereas most changes in AUTOSAR are extensions or clarifications not causing any issues for backward compatibility, changes to the interfaces need to be reflected in the user configuration. These changes are described in the following sub chapters.

2.6.1 Client Server Interfaces

The client server interfaces differ between the AUTOSAR versions. The main differences are typically the change of the parameter type or differences in the return values. Since the interfaces between the Client and the Server needs to be compatible but not identical, the Csm can provide a dedicated interface for each AUTOSAR version introducing a change to the interface. Please use the configuration parameters

> CsmKeySwcInterfaceVersion at a CsmKey

> Csm<Service>SwcInterfaceVersion at a CsmPrimitive

> CsmCallbackAsrVersion at a CsmCallback

to configure the required AUTOSAR version of the correspdoning service port. Furthermore, these parameters give a detailled description about the differences of the correspoding interfaces in the different AUTOSAR versions.

Note
The Csm will provide the Client Server Interfaces postfixed by the AUTOSAR version. If a non-postfixed version of the interface is required, Csm can provide one version without postfix in addition by setting the CsmSwcMainInterfaceVersion.

Caution
Even if the interfaces declare different return values for the operations, the Csm always returns the values received from the Crypto Driver. There is no filter or mapping mechanism implemented as this would increase the runtime and could hide potentially needed extensions to a return value of a Crypto Driver (e.g. a hardware dependent return value).

If the connected SWC cannot handle other values then specified, a wrapper SWC needs to be used to filter/map the return values before passing it to the target SWC.

Note
The Csm provides several APIs with suffixes

- DataRef e.g. Csm_HashDataRef
- Asr4_03, AsrR19-11, … e.g. Csm_MacGenerateAsr4_03
- NoOpMode e.g. Csm_CancelJobNoOpMode

These APIs are used only by RTE/SWCs to be able to provide the Client Server Interfaces in different AUTOSAR versions. These API functions wrap existing API functions and do not add functionality.

2.6.2 Csm Job Type Layout

The Csm_JobType varies in the different AUTOSAR versions. This implementation of the Csm provides a configurable super set of the type to enable the usage of Crypto Drivers implementing different AUTOSAR versions. However, the type should be kept as small as possible to save memory and runtime resources. Use the configuration parameters located in the `CsmGeneral/CsmJobTypeLayout` container to disable not needed fields.

2.6.3 BSW Callbacks

The callbacks used for asynchronous jobs have been changed in the different AUTOSAR versions. The concrete version can be set per callback using the CsmCallbackAsrVersion parameter of the CsmCallback. Please refer to the parameter or chapter 4.5 for the details of the different versions.

2.6.4 KeyId Parameter in Job Key Primitives

Introduced with [3], the Job Key Primitives (e.g., Csm_JobKeySetValid) all had a keyId parameter. This parameter was removed in [6] without renaming the API. However, as there is the potential need to have the former or both variants available, this implementation provides the former declarations specified in [3] suffixed by 44x, e.g., Csm_JobKeySetValid44x.

By setting the parameter `CsmDefaultJobKeyApiVersion` in the `CsmGeneral/CsmBswApiCompatibility` container, the default declaration of the API can be set.

Basic Knowledge
Depending on the version, the Csm provides preprocessor definitions to let the consuming application call the former API, e.g.,

```
#if (CSM_JOB_KEY_API_VERSION == CSM_ASR_VERSION_4_04)
# define Csm_JobKeySetValid Csm_JobKeySetValid44x
#endif
```

This also works for function pointers as only the name is replaced.

Furthermore, there is a mechanism to set the version on a per user base, e.g. for each source file that is including the Csm header. This mechanism can be used if the source file provides a unique preprocessor definition identifying the source before including the Csm header. So, the Csm can decide which version it provides to the including source file. This behavior can be configured in the

`CsmGeneral/CsmBswApiCompatibility/CsmBswUserConfiguration`

container.

Example
Example.c:

```
#define EXAMPLE_SOURCE
#include "Csm.h"
…
```

Through this, the Csm can decide which API version shall be provided:

Csm_Cfg.h:

```
#if defined (EXAMPLE_SOURCE)
# define CSM_JOB_KEY_API_VERSION CSM_ASR_VERSION_4_04
#elif defined (MIP_SOURCE)
# define CSM_JOB_KEY_API_VERSION CSM_ASR_VERSION_R21_11
#else
# define CSM_JOB_KEY_API_VERSION CSM_ASR_VERSION_R21_11
#endif
```

2.7 Multicore / Partitioning

The Csm supports partitioning by each queue as specified in [6].

2.7.1 Initialization

Each partition calls the CSM_Init and initializes their own artefacts.

The module generates dedicated `Csm_InitMemory_<OsApplicationName>()` functions for each partition which take care of initializing the partition dependent init variables to 0. Initializing can be done in the following ways:

- Startup core calls all available `Csm_InitMemory_<OsApplicationName>()` functions.
  - This requires that the startup core can write to the memory of all partitions.
  - This might require mapping the partition specific init variables to shared memory.
  - This would require a dedicated MPU region for the init variables of the partitions.
- Each core calls the `Csm_InitMemory_<OsApplicationName>()` function for its partitions.
  - This can be used when init variables shall be mapped to core local memory which might not be accessible from other cores.
- Do not call the `Csm_InitMemory_<OsApplicationName>()` at all when ZERO_INIT variables are already correctly initialized.
  - Initialization is usually done by vLinkGen or Compiler specific startup code.

Expert Knowledge
Typically, `Csm_InitMemory()` is called before `OsStart()` so that MPU is not configured yet.

2.8 Interrupt Context

When using a Crypto Driver with Hardware support (e.g. HSM), the Csm_CallbackNotification (see Chapter 4.4) may be called in the context of an interrupt (ISR). As a result, the application Callback function (see Chapter 4.5) will be called in the same context. Hence, the application must ensure that the provided Callback function is implemented accordingly. This includes matching the system needs for runtime requirements and not invoking functions which may not be called in an ISR.

Basic Knowledge
This is not true for callbacks to RTE/SWC as their invocation will be deferred to the Csm_MainFunction.

3 Integration

This chapter gives necessary information for the integration of the MICROSAR Classic CSM into an application environment of an ECU.

3.1 Scope of Delivery

The delivery of the CSM contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Description
Csm.c	Source file of the CSM.
Csm_Rte.c	Source file of the CSM providing interfaces used by RTE.
Csm.h	Header file of the CSM providing all service declarations.
Csm_Rte.h	Header file of the CSM describing interfaces used by RTE.
Csm_Cbk.h	Header file which contains the callback function declaration.
Csm_Types.h	Deprecated types header. Replaced by Crypto_GeneralTypes.h
Crypto_GeneralTypes.h	Common header file for types used within the crypto stack.

Table 3-1 Static files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator 5 Pro.

File Name	Description
Csm_Cfg.c	Configuration source file.
Csm_Cfg.h	Configuration header file.
Csm_Generated_Types.h	Configuration header file for Crypto_GeneralTypes.h.

Table 3-2 Generated files

3.2 Critical Sections

The CSM has critical code sections which must not be interrupted. The sections are related to the queue handling, job sorting and job queuing.

The CSM module calls the following function when entering a critical section:

> SchM_Enter_Csm_CSM_EXCLUSIVE_AREA_0

When the critical section is left the following function is called by the CSM module:

> SchM_Exit_Csm_CSM_EXCLUSIVE_AREA_0

This critical section is needed to ensure consistency of global RAM variables regarding the queue handling.

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Type Definitions

Application related types defined by the CSM are described in this chapter. All types not described here are defined by the CSM as described in [1].

Type Name	C-Type	Description	Value Range
Crypto_OperationModeType	uint8	Indicator of the mode(s)/operation(s) to be performed.	`CRYPTO_OPERATIONMODE_START` Perform start operation
			`CRYPTO_OPERATIONMODE_UPDATE` Perform update operation
			`CRYPTO_OPERATIONMODE_FINISH` Perform finish operation
			`CRYPTO_OPERATIONMODE_STREAMSTART` Perform start and update operation
			`CRYPTO_OPERATIONMODE_SINGLECALL` Perform start, update and finish operation
			`CRYPTO_OPERATIONMODE_SAVE_CONTEXT` Perform save workspace context
			`CRYPTO_OPERATIONMODE_RESTORE_CONTEXT` Perform restore workspace context
Crypto_VerifyResultType	uint8	Enumeration of the result type of verification operations.	`CRYPTO_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"
			`CRYPTO_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".

Table 4-1 Type definitions

4.2 Services provided by CSM

4.2.1 Csm_Init

Prototype	
`void Csm_Init (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Initializes the CSM module. Called by each partition.	
Particularities and Limitations	
Set all service states to initial idle.	
Call context	
> TASK > This function is Synchronous > This function is non-reentrant	

Table 4-2 Csm_Init

4.2.2 Csm_InitMemory

Prototype	
`void Csm_InitMemory (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Power-up memory initialization. Called by each partition.	
Particularities and Limitations	
Use this function in case these variables are not initialized by the startup code. Module is uninitialized. Initialize component variables at power up.	
Call context	
> TASK > This function is Synchronous > This function is non-reentrant	

Table 4-3 Csm_InitMemory

4.2.3 Csm_MainFunction

Prototype	
`void Csm_MainFunction ( void )`	
Parameter	
void	none
Return code	
void	none
Functional Description	
API to be called cyclically to process the requested jobs.	
Particularities and Limitations	
> Declared and called by SchM > Configured and generated for each partition > Is called cyclically.	
Call context	
> TASK > This function is synchronous > This function is non-reentrant	

Table 4-4 Csm_MainFunction

4.2.4 Csm_GetVersionInfo

Prototype	
`void Csm_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer where the version info data shall be stored.
Return code	
void	none
Functional Description	
Returns the version information of this module.	
Particularities and Limitations	
Stores version information, i.e. Module Id, Vendor Id, Vendor specific version numbers to structure pointed by versioninfo. GetVersionInfo API is enabled via pre-compile configuration.	
Call context	
> TASK > This function is Synchronous > This function is reentrant	

Table 4-5 Csm_GetVersionInfo

4.2.5 Csm_CancelJob

Prototype	
`Std_ReturnType` **`Csm_CancelJob`** `(uint32 jobId, Crypto_OperationModeType mode)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Not used, just for interface compatibility provided
Return code	
Std_ReturnType	E_OK Request successful – Job was cancelled or was already Idle
Std_ReturnType	E_NOT_OK Request failed – Job could not be cancelled
Std_ReturnType	CRYPTO_E_JOB_CANCELED Request pending - Job will be cancelled with next callback notification
Functional Description	
Cancels the job processing of asynchronous or streaming jobs. For cancellation of asynchronous jobs the application must be able to handle two callbacks. It is possible that the normal job callback notification will be called while cancelling. The cancel callback notification can be identified by contained result CRYPTO_E_JOB_CANCELED.	
Particularities and Limitations	
Removes the job in the Csm Queue and calls the job's callback with the result CRYPTO_E_JOB_CANCELED. It also passes the cancellation command to the CryIf to try to cancel the job in the Crypto Driver.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different jobId	

Table 4-6 Csm_CancelJob

4.2.6 Csm_KeyElementSet

Prototype	
`Std_ReturnType` **`Csm_KeyElementSet`** `(uint32 keyId, uint32 keyElementId, const uint8 *keyPtr, uint32 keyLength)`	
Parameter	
keyId [in]	Holds the identifier of the key using the CSM service.
keyElementId [in]	Holds the identifier of the key element to be written.
keyPtr [in]	Holds the pointer to the key element bytes to be processed.
keyLength [in]	Contains the number of key element bytes.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy
	CRYPTO_E_KEY_WRITE_FAIL Request failed because write access was denied
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed because the key is not available
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed because key element size does not match size of provided data
Functional Description	
Sets the given key element bytes to the key identified by keyId.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-7 Csm_KeyElementSet

4.2.7 Csm_KeySetValid

Prototype	
`Std_ReturnType` **`Csm_KeySetValid`** `(uint32 keyId)`	
Parameter	
keyId [in]	Holds the identifier of the key for which a new material shall be validated.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy
Functional Description	
Sets the key state of the key identified by keyId to valid.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-8 Csm_KeySetValid

4.2.8 Csm_KeySetInvalid

Prototype	
`Std_ReturnType` **`Csm_KeySetInvalid`** `(uint32 keyId)`	
Parameter	
keyId [in]	Holds the identifier of the key for which a new material shall be invalidated.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy
Functional Description	
Sets the key status to invalid. The key cannot be used any longer for cryptographic operations until it has been set to valid state again.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-9 Csm_KeySetInvalid

4.2.9 Csm_KeyGetStatus

Prototype	
`Std_ReturnType` **`Csm_KeyGetStatus`** `(uint32 keyId, Crypto_KeyStatusType *keyStatusPtr)`	
Parameter	
keyId [in]	Holds the identifier of the key for which a new material shall be invalidated.
keyStatusPtr [out]	Contains the pointer to the data where the status of the key shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
Functional Description	
Returns the key state of the key identified by keyId.	
Particularities and Limitations	
Returns the key state of the key identified by keyId.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-10 Csm_KeyGetStatus

4.2.10 Csm_KeyElementGet

Prototype	
`Std_ReturnType` **`Csm_KeyElementGet`** `(uint32 keyId, uint32 keyElementId, uint8 *keyPtr, uint32 *keyLengthPtr)`	
Parameter	
keyId [in]	Holds the identifier of the key from which a key element shall be extracted.
keyElementId [in]	Holds the identifier of the key element to be extracted.
keyPtr [out]	Holds the pointer to the memory location where the key shall be copied to.
inout] [out]	keyLengthPtr Holds a pointer to the memory location in which the output buffer length in bytes is stored. On calling this function, this parameter shall contain the buffer length in bytes of the keyPtr. When the request has finished, the actual size of the written input bytes shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy
	CRYPTO_E_KEY_READ_FAIL Request failed because read access was denied
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed because the key is not available
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Retrieves the key element bytes from a specific key element of the key identified by the keyId and stores the key element in the memory location pointed by the key pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-11 Csm_KeyElementGet

4.2.11 Csm_KeyElementCopy

Prototype	
`Std_ReturnType` **`Csm_KeyElementCopy`** `(uint32 keyId, uint32 keyElementId, uint32 targetKeyId, uint32 targetKeyElementId)`	
Parameter	
keyId [in]	Holds the identifier of the key whose key element shall be the source element.
keyElementId [in]	Holds the identifier of the key element which shall be the source for the copy operation.
targetKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
targetKeyElementId [in]	Holds the identifier of the key element which shall be the destination for the copy operation.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
	CRYPTO_E_KEY_READ_FAIL Request failed because read access was denied
	CRYPTO_E_KEY_WRITE_FAIL Request failed because write access was denied
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed because the key is not available
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed because key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
This function shall copy a key elements from one key to a target key.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-12 Csm_KeyElementCopy

4.2.12 Csm_KeyElementCopyPartial

Prototype	
`Std_ReturnType` **`Csm_KeyElementCopyPartial`** `(uint32 keyId, uint32 keyElementId, uint32 keyElementSourceOffset, uint32 keyElementTargetOffset, uint32 keyElementCopyLength, uint32 targetKeyId, uint32 targetKeyElementId)`	
Parameter	
keyId [in]	Holds the identifier of the key whose key element shall be the source element.
keyElementId [in]	Holds the identifier of the key element which shall be the source for the copy operation.
keyElementSourceOffset[in]	This is the offset of the source key element indicating the start index of the copy operation.
keyElementTargetOffset[in]	This is the offset of the destination key element indicating the start index of the copy operation.
keyElementCopyLength[in]	Specifies the number of bytes that shall be copied.
targetKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
targetKeyElementId [in]	Holds the identifier of the key element which shall be the destination for the copy operation.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
	CRYPTO_E_KEY_READ_FAIL Request failed because read access was denied
	CRYPTO_E_KEY_WRITE_FAIL Request failed because write access was denied
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed because the key is not available
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed because key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element.
Functional Description	
This function shall copy a key elements from one key to a target key partially.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-13 Csm_KeyElementCopyPartial

4.2.13 Csm_KeyCopy

Prototype	
`Std_ReturnType **Csm_KeyCopy** (uint32 keyId, uint32 targetKeyId)`	
Parameter	
keyId [in]	Holds the identifier of the key whose key element shall be the source element.
targetKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
	CRYPTO_E_KEY_READ_FAIL Request failed because read access was denied
	CRYPTO_E_KEY_WRITE_FAIL Request failed because write access was denied
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed because the key is not available
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed because key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
This function shall copy all key elements from the source key to a target key.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-14 Csm_KeyCopy

4.2.14 Csm_RandomSeed

Prototype	
`Std_ReturnType` **`Csm_RandomSeed`** `(uint32 keyId, const uint8 *seedPtr, uint32 seedLength)`	
Parameter	
keyId [in]	Holds the identifier of the key for which a new material shall be generated.
seedPtr [in]	Holds a pointer to the memory location containing the data to feed the seed.
seedLength [in]	Contains the length of the seed in bytes.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
Functional Description	
Feeds the key element CRYPTO_KE_RANDOM_SEED with a random seed.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-15 Csm_RandomSeed

4.2.15 Csm_KeyGenerate

Prototype	
`Std_ReturnType **Csm_KeyGenerate** (uint32 keyId)`	
Parameter	
keyId [in]	Holds the identifier of the key for which a new material shall be generated.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Generates new key material and store it in the key identified by keyId.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-16 Csm_KeyGenerate

4.2.16 Csm_KeyDerive

Prototype	
`Std_ReturnType` **`Csm_KeyDerive`** `(uint32 keyId, uint32 targetKeyId)`	
Parameter	
keyId [in]	Holds the identifier of the key used for key derivation.
targetKeyId [in]	Holds the identifier of the key which is used to store the derived key.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
	CRYPTO_E_KEY_READ_FAIL Request failed, not allowed to extract key element
	CRYPTO_E_KEY_WRITE_FAIL Request failed, not allowed to write key element
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Derives a new key by using the key elements in the given key identified by the keyId. The given key contains the key elements for the password and salt. The derived key is stored in the key element with the id 1 of the key identified by targetCryptoKeyId.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK	
> This function is Synchronous	
> This function is reentrant for different keyId	

Table 4-17 Csm_KeyDerive

4.2.17 Csm_KeyExchangeCalcPubVal

Prototype	
`Std_ReturnType` **`Csm_KeyExchangeCalcPubVal`** `(uint32 keyId, uint8 *publicValuePtr, uint32 *publicValueLengthPtr)`	
Parameter	
keyId [in]	Holds the key identifier of the key to be used for the key exchange protocol.
publicValuePtr [out]	Contains the pointer to the data where the public value shall be stored.
inout] [out]	publicValueLengthPtr Holds a pointer to the memory location in which the public value length in bytes is stored. On calling this function, this parameter shall contain the size of the buffer in bytes provided by publicValuePtr. When the request has finished, the actual length of the returned value shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_KEY_NOT_VALID Request failed because the key's state is "invalid"
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Calculates the public value of the current user for the key exchange and stores the public key in the memory location pointed by the public value pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-18 Csm_KeyExchangeCalcPubVal

4.2.18 Csm_KeyExchangeCalcSecret

Prototype	
`Std_ReturnType` **`Csm_KeyExchangeCalcSecret`** `(uint32 keyId, const uint8 *partnerPublicValuePtr, uint32 partnerPublicValueLength)`	
Parameter	
keyId [in]	Holds the key identifier of the key to be used for the key exchange protocol.
partnerPublicValuePtr [in]	Holds the pointer to the memory location containing the partner's public value.
partnerPublicValueLength [in]	Contains the number of bytes of the partner public value.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Calculates the shared secret key for the key exchange with the key material of the key identified by the keyId and the partner public key. The shared secret key is stored as a key element in the same key.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-19 Csm_KeyExchangeCalcSecret

4.2.19 Csm_CertificateParse

Prototype	
`Std_ReturnType` **`Csm_CertificateParse`** `(uint32 keyId)`	
Parameter	
keyId [in]	Holds the identifier of the key to be used for the certificate parsing.
Return code	
Std_ReturnType	E_OK Request successful
Std_ReturnType	E_NOT_OK Request failed
Std_ReturnType	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
Functional Description	
This function shall dispatch the certificate parse function to the CryIf.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-20 Csm_CertificateParse

4.2.20 Csm_CertificateVerify

Prototype	
`Std_ReturnType` **`Csm_CertificateVerify`** `(uint32 keyId, uint32 verifyKeyId, Crypto_VerifyResultType *verifyPtr)`	
Parameter	
keyId [in]	Holds the identifier of the key to be used to validate the certificate.
verifyKeyId [in]	Holds the identifier of the key containing the certificate to be verified.
verifyPtr [out]	Holds a pointer to the memory location, which will contain the result of the certificate verification.
Return code	
Std_ReturnType	E_OK Request successful
Std_ReturnType	E_NOT_OK Request failed
Std_ReturnType	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
Functional Description	
Verifies the certificate stored in the key referenced by verifyKeyId with the certificate stored in the key referenced by keyId. Note: Only certificates stored in the same Crypto Driver can be verified against each other. If the key element CRYPTO_KE_CERTIFICATE_CURRENT_TIME is used for the verification of the validity period of the certificate identified by verifyKeyId, it shall have the same format as the timestamp in the certificate.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is Synchronous > This function is reentrant for different keyId	

Table 4-21 Csm_CertificateVerify

4.2.21 Csm_Hash

Prototype	
`Std_ReturnType **Csm_Hash** (uint32 jobId, Crypto_OperationModeType mode, const uint8 *dataPtr, uint32 dataLength, uint8 *resultPtr, uint32 *resultLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
dataPtr [in]	Contains the pointer to the data for which the hash shall be computed.
dataLength [in]	Contains the number of bytes to be hashed.
resultPtr [out]	Contains the pointer to the data where the hash value shall be stored.
resultLengthPtr [in,out]	Holds a pointer to the memory location in which the output length in bytes 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.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Uses the given data to perform the hash calculation and stores the hash.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-22 Csm_Hash

4.2.22 Csm_MacGenerate

Prototype	
`Std_ReturnType **Csm_MacGenerate** (uint32 jobId, Crypto_OperationModeType mode, const uint8 *dataPtr, uint32 dataLength, uint8 *macPtr, uint32 *macLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
dataPtr [in]	Contains the pointer to the data for which the MAC shall be computed.
dataLength [in]	Contains the number of bytes for the MAC generation.
macPtr [out]	Contains the pointer to the data where the MAC shall be stored.
macLengthPtr [in,out]	Holds a pointer to the memory location in which the output length in bytes is stored. On calling this function, this parameter shall contain the size of the buffer provided by macPtr. When the request has finished, the actual length of the returned MAC shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Uses the given data to perform a MAC generation and stores the MAC in the memory location pointed to by the MAC pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-23 Csm_MacGenerate

4.2.23 Csm_MacVerify

Prototype	
`Std_ReturnType` **`Csm_MacVerify`** `(uint32 jobId, Crypto_OperationModeType mode, const uint8 *dataPtr, uint32 dataLength, const uint8 *macPtr, uint32 macLength, Crypto_VerifyResultType *verifyPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
dataPtr [in]	Contains the pointer to the data for which the MAC shall be verified.
dataLength [in]	Contains the number of data bytes for which the MAC shall be verified.
macPtr [in]	Holds a pointer to the MAC to be verified.
macLength [in]	Contains the MAC length in BITS to be verified.
verifyPtr [out]	Holds a pointer to the memory location, which will hold the result of the MAC verification.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Verifies the given MAC by comparing if the MAC is generated with the given data.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-24 Csm_MacVerify

4.2.24 Csm_Encrypt

Prototype	
`Std_ReturnType` **`Csm_Encrypt`** `(uint32 jobId, Crypto_OperationModeType mode, const uint8 *dataPtr, uint32 dataLength, uint8 *resultPtr, uint32 *resultLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
dataPtr [in]	Contains the pointer to the data to be encrypted.
dataLength [in]	Contains the number of bytes to encrypt.
resultPtr [out]	Contains the pointer to the data where the encrypted data shall be stored.
resultLengthPtr [in,out]	Holds a pointer to the memory location in which the output length information is stored in bytes. 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.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Encrypts the given data and stores the ciphertext in the memory location pointed by the result pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-25 Csm_Encrypt

4.2.25 Csm_Decrypt

Prototype	
`Std_ReturnType Csm_Decrypt (uint32 jobId, Crypto_OperationModeType mode, const uint8 *dataPtr, uint32 dataLength, uint8 *resultPtr, uint32 *resultLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
dataPtr [in]	Contains the pointer to the data to be decrypted.
dataLength [in]	Contains the number of bytes to decrypt.
resultPtr [out]	Contains the pointer to the data where the decrypted data shall be stored.
resultLengthPtr [in,out]	Holds a pointer to the memory location in which the output length information is stored in bytes. 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.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Decrypts the given encrypted data and stores the decrypted plaintext in the memory location pointed by the result pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-26 Csm_Decrypt

4.2.26 Csm_AEADEncrypt

Prototype	
`Std_ReturnType **Csm_AEADEncrypt** (uint32 jobId, Crypto_OperationModeType mode, const uint8 *plaintextPtr, uint32 plaintextLength, const uint8 *associatedDataPtr, uint32 associatedDataLength, uint8 *ciphertextPtr, uint32 *ciphertextLengthPtr, uint8 *tagPtr, uint32 *tagLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
plaintextPtr [in]	Contains the pointer to the data to be encrypted.
plaintextLength [in]	Contains the number of bytes to encrypt.
associatedDataPtr [in]	Contains the pointer to the associated data.
associatedDataLength [in]	Contains the number of bytes of the associated data.
ciphertextPtr [out]	Contains the pointer to the data where the encrypted data shall be stored.
ciphertextLengthPtr [in,out]	Holds a pointer to the memory location in which the output length in bytes of the ciphertext is stored. On calling this function, this parameter shall contain the size of the buffer in bytes provided by ciphertextPtr. When the request has finished, the actual length of the returned value shall be stored.
tagPtr [out]	Contains the pointer to the data where the Tag shall be stored.
tagLengthPtr [in,out]	Holds a pointer to the memory location in which the output length in bytes of the Tag is stored. On calling this function, this parameter shall contain the size of the buffer in bytes provided by tagPtr. When the request has finished, the actual length of the returned value shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Uses the given input data to perform a AEAD encryption and stores the ciphertext and the MAC in the memory locations pointed by the ciphertext pointer and Tag pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-27 Csm_AEADEncrypt

4.2.27 Csm_AEADDecrypt

Prototype	
`Std_ReturnType` **`Csm_AEADDecrypt`** `(uint32 jobId, Crypto_OperationModeType mode, const uint8 *ciphertextPtr, uint32 ciphertextLength, const uint8 *associatedDataPtr, uint32 associatedDataLength, const uint8 *tagPtr, uint32 tagLength, uint8 *plaintextPtr, uint32 *plaintextLengthPtr, Crypto__VerifyResultType *verifyPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
ciphertextPtr [in]	Contains the pointer to the data to be decrypted.
ciphertextLength [in]	Contains the number of bytes to decrypt.
associatedDataPtr [in]	Contains the pointer to the associated data.
associatedDataLength [in]	Contains the number of bytes of the associated data.
tagPtr [in]	Contains the pointer to the data where the Tag shall be stored.
tagLength [in]	Contains the length in bytes of the Tag to be verified.
plaintextPtr [out]	Contains the pointer to the data where the encrypted data shall be stored.
plaintextLengthPtr [in,out]	Holds a pointer to the memory location in which the output length in bytes of the plaintext is stored. On calling this function, this parameter shall contain the size of the buffer in bytes provided by plaintextPtr. When the request has finished, the actual length of the returned value shall be stored.
verifyPtr [out]	Contains the pointer to the result of the verification.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Uses the given data to perform an AEAD decryption and stores the plaintext and the result of in the memory locations pointed by the plaintext pointer and verifyPtr pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-28 Csm_AEADDecrypt

4.2.28 Csm_SignatureGenerate

Prototype	
`Std_ReturnType` **`Csm_SignatureGenerate`** `(uint32 jobId, Crypto_OperationModeType mode, const uint8 *dataPtr, uint32 dataLength, uint8 *resultPtr, uint32 *resultLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
dataPtr [in]	Contains the pointer to the data to be signed.
dataLength [in]	Contains the number of bytes to sign.
resultPtr [out]	Contains the pointer to the data where the signature shall be stored.
resultLengthPtr [in,out]	Holds a pointer to the memory location in which the output length in bytes is stored. On calling this function, this parameter shall contain the size of the buffer in bytes provided by resultPtr. When the request has finished, the actual length of the returned value shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Uses the given data to perform the signature calculation and stores the signature in the memory location pointed by the result pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-29 Csm_SignatureGenerate

4.2.29 Csm_SignatureVerify

Prototype	
`Std_ReturnType` **`Csm_SignatureVerify`** `(uint32 jobId, Crypto_OperationModeType mode, const uint8 *dataPtr, uint32 dataLength, const uint8 *signaturePtr, uint32 signatureLength, Crypto_VerifyResultType *verifyPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
mode [in]	Indicates which operation mode(s) to perfom.
dataPtr [in]	Contains the pointer to the data to be verified.
dataLength [in]	Contains the number of bytes to be verified.
signaturePtr [in]	Holds a pointer to the signature to be verified.
signatureLength [in]	Contains the signature length in bytes.
verifyPtr [out]	Holds a pointer to the memory location, which will hold the result of the signature verification.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Verifies the given MAC by comparing if the signature is generated with the given data.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-30 Csm_SignatureVerify

4.2.30 Csm_RandomGenerate

Prototype	
`Std_ReturnType` **`Csm_RandomGenerate`** `(uint32 jobId, uint8 *resultPtr, uint32 *resultLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
resultPtr [out]	Holds a pointer to the memory location which will hold the result of the random number generation.
resultLengthPtr [in,out]	Holds a pointer to the memory location in which the result length in bytes is stored. On calling this function, this parameter shall contain the number of random bytes, which shall be stored to the buffer provided by resultPtr. When the request has finished, the actual length of the returned value shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_ENTROPY_EXHAUSTION Request failed, entropy of random number generator is exhausted
	CRYPTO_E_SMALL_BUFFER The provided buffer is too small to store the result
Functional Description	
Generate a random number and stores it in the memory location pointed by the result pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-31 Csm_RandomGenerate

4.2.31 Csm_JobKeySetValid

Prototype	
`Std_ReturnType` **`Csm_JobKeySetValid`** `(uint32 jobId)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
Return code	
Std_ReturnType	E_OK: Request successful
	E_NOT_OK: Request failed
	CRYPTO_E_BUSY: Request failed, service is busy or queue is full
Functional Description	
Stores the key if necessary and sets the key state of the key configured in the job to valid.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-32 Csm_JobKeySetValid

4.2.32 Csm_JobKeySetInvalid

Prototype	
`Std_ReturnType` **`Csm_JobKeySetInvalid`** `(uint32 jobId)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
Return code	
Std_ReturnType	E_OK: Request successful
	E_NOT_OK: Request failed
	CRYPTO_E_BUSY: Request failed, service is busy or queue is full
Functional Description	
Sets the key status to invalid. The key cannot be used any longer for cryptographic operations until it has been set to valid state again.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-33 Csm_JobKeySetInvalid

4.2.33 Csm_JobRandomSeed

Prototype	
`Std_ReturnType` **`Csm_JobRandomSeed`** `(uint32 jobId, const uint8 *seedPtr, uint32 seedLength)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
seedPtr [in]	Holds a pointer to the memory location which contains the data to feed the seed.
seedLength [in]	Contains the length of the seed in bytes.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
Functional Description	
This function shall dispatch the random seed function to the configured Crypto Driver object.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-34 Csm_JobRandomSeed

4.2.34 Csm_JobKeyGenerate

Prototype	
`Std_ReturnType` **`Csm_JobKeyGenerate`** `(uint32 jobId,)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Generates new key material and stores it in the key configured in the job.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-35 Csm_JobKeyGenerate

4.2.35 Csm_JobKeyDerive

Prototype	
`Std_ReturnType` **`Csm_JobKeyDerive`** `(uint32 jobId, uint32 targetKeyId)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
targetKeyId [in]	Holds the identifier of the key which is used to store the derived key.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_READ_FAIL Request failed, not allowed to extract key element
	CRYPTO_E_KEY_WRITE_FAIL Request failed, not allowed to write key element
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, key element sizes are not compatible
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Derives a new key by using the key elements in the key configured in the job. The key contains the key elements for the password and salt. The derived key is stored in the key element with the id 1 of the key identified by targetKeyId.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-36 Csm_JobKeyDerive

4.2.36 Csm_JobKeyExchangeCalcPubVal

Prototype	
`Std_ReturnType` **`Csm_JobKeyExchangeCalcPubVal`** `(uint32 jobId, uint8 *publicValuePtr, uint32 *publicValueLengthPtr)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
publicValueLengthPtr [in, out]	Holds a pointer to the memory location in which the public value length information is stored. On calling this function, this parameter shall contain the size of the buffer provided by publicValuePtr. When the request has finished, the actual length of the returned value shall be stored.
publicValuePtr [out]	Contains the pointer to the data where the public value shall be stored.
Return code	
Std_ReturnType	E_OK Request successful
	E_NOT_OK Request failed
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element
Functional Description	
Calculates the public value of the current user for the key exchange and stores the public key in the memory location pointed by the public value pointer.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-37 Csm_JobKeyExchangeCalcPubVal

4.2.37 Csm_JobKeyExchangeCalcSecret

Prototype		
`Std_ReturnType` **`Csm_JobKeyExchangeCalcSecret`** `(uint32 jobId, const uint8 *partnerPublicValuePtr, uint32 partnerPublicValueLength)`		
Parameter		
jobId [in]	Holds the identifier of the job using the CSM service.	
partnerPublicValuePtr [in]	Holds the pointer to the memory location which contains the partner's public value.	
partnerPublicValueLength [in]	Contains the length of the partner's public value in bytes.	
Return code		
Std_ReturnType	E_OK Request successful	
	E_NOT_OK Request failed	
	CRYPTO_E_BUSY Request failed, service is busy or queue is full	
	CRYPTO_E_KEY_NOT_VALID Request failed, the key's state is "invalid"	
	CRYPTO_E_KEY_EMPTY Request failed because of uninitialized source key element	
Functional Description		
Calculates the shared secret key for the key exchange with the key material of the key configured in the job and the partner public key. The shared secret key is stored as a key element in the same key.		
Particularities and Limitations		
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.		
Call context		
> TASK > This function is reentrant for different jobId		

Table 4-38 Csm_JobKeyExchangeCalcSecret

4.2.38 Csm_SaveContextJob

Prototype		
`Std_ReturnType **Csm_SaveContextJob** (uint32 jobId, uint8 *contextBufferPtr, uint32 *contextBufferLengthPtr)`		
Parameter		
jobId [in]	Holds the identifier of the job using the CSM service.	
contextBufferPtr [out]	Pointer to the buffer in the application where the context data shall be stored to.	
contextBufferLengthPtr [in,out]	Pointer to the buffer, where the length value is located.As input data it provides the maximum length of data available in contextBufferPtr.As output data it provides the actual length of data located in context BufferPtr(or 0 in case of a failure)	
Return code		
Std_ReturnType	E_OK Context data successfully provided.	
	E_NOT_OK Context data could not be provided, values are not valid.	
	CRYPTO_E_BUSY Request failed, service is busy or queue is full	
Functional Description		
The Crypto Driver stores the internal context of the respective crypto operation to the contextBuffer.		
Particularities and Limitations		
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.		
Call context		
> TASK > This function is reentrant for different jobId		

Table 4-39 Csm_SaveContextJob

4.2.39 Csm_RestoreContextJob

Prototype	
`Std_ReturnType` **`Csm_ RestoreContextJob`** `(uint32 jobId, uint8 *contextBufferPtr, uint32 contextBufferLength)`	
Parameter	
jobId [in]	Holds the identifier of the job using the CSM service.
contextBufferPtr [in]	Pointer to the buffer, where the context data are located that shall be restored.
contextBufferLength [in]	Provides the length of context data that are located in contextBufferPtr.
Return code	
Std_ReturnType	E_OK Context data successfully restored.
	E_NOT_OK Context data could not be restored.
	CRYPTO_E_BUSY Request failed, service is busy or queue is full
Functional Description	
The Crypto Driver extracts the context data from the contextBuffer and restores the internal state so that further crypto operation of this crypto service will continue at the exact point when the context was taken.	
Particularities and Limitations	
The returned value is defined by the underlying Crypto Driver and may differ from the listed ones.	
Call context	
> TASK > This function is reentrant for different jobId	

Table 4-40 Csm_RestoreContextJob

4.3 Services used by CSM

In the following table services provided by other components, which are used by the CSM are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
CryIf	CryIf_CancelJob
CryIf	CryIf_CertificateParse
CryIf	CryIf_CertificateVerify
CryIf	CryIf_KeyDerive
CryIf	CryIf_KeyElementCopy
CryIf	CryIf_KeyElementCopyPartial
CryIf	CryIf_KeyElementGet
CryIf	CryIf_KeyElementSet
CryIf	CryIf_KeyExchangeCalcPubVal
CryIf	CryIf_KeyExchangeCalcSecret
CryIf	CryIf_KeyGenerate
CryIf	CryIf_KeyGetStatus
CryIf	CryIf_KeySetInvalid
CryIf	CryIf_KeySetValid
CryIf	CryIf_ProcessJob
DET	Det_ReportError
DET	Det_ReportRuntimeError

Table 4-41 Services used by the CSM

4.4 Callback Functions Provided by CSM

This chapter describes the callback functions that are implemented by the CSM and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `Csm_Cbk.h` by the CSM.

4.4.1 Csm_CallbackNotification

Prototype	
`void` **`Csm_CallbackNotification`** `( Crypto_JobType *job, Crypto_ResultType result )`	
Parameter	
`job`	Points to the completed job's information structure. It contains a callbackID to identify which job is finished.
`result`	Contains the result of the cryptographic operation. The second most important bit of result must not be set. It must always be zero.
Return code	
`void`	none
Functional Description	
Notifies the CSM about the completion of the request with the result of the cryptographic operation.	
Particularities and Limitations	
> TASK | ISR2 | ISR1 > This function is synchronous. > This function is non-reentrant.	

Table 4-42 Csm_CallbackNotification

Caution
The second most important bit of parameter `result` of function **`Csm_CallbackNotification`** must not be set. It must always be zero.

4.5 Callback Functions Invoked by CSM

This chapter describes the callback functions that are implemented by other BSW modules and that can be invoked by the CSM. The signature can be individually configured per callback function via the Cfg5.

Caution
The following callbacks invoked by CSM might occur in the context of the caller of the asynchonous jobs. Thus, change in the caller to the state waiting for the callback **before** calling the corresponding CSM API function with an asynchronous job.

4.5.1 CallbackFunc ASR 4.3

Prototype	
`void` **`<CallbackFunc>`** `( Crypto_JobType *job, Std_ReturnType result )`	
Parameter	
`job`	Points to the completed job's information structure. It contains a callbackID to identify which job is finished.
`result`	Contains the result of the cryptographic operation.
Return code	
`void`	none
Functional Description	
Notifies the CSM about the completion of the request with the result of the cryptographic operation.	
Particularities and Limitations	
> TASK | ISR2 | ISR1 > This function is synchronous. > This function is non-reentrant.	

Table 4-43 CallbackFunc ASR 4.3

4.5.2 CallbackFunc ASR 4.4

Prototype	
`void` **`<CallbackFunc>`** `( const uint32 jobID, Csm_ResultType result )`	
Parameter	
`jobID`	Identifies the job which is finished and caused the callback.
`result`	Contains the result of the cryptographic operation.
Return code	
`void`	none

Functional Description
Notifies the CSM about the completion of the request with the result of the cryptographic operation.
Particularities and Limitations
> TASK | ISR2 | ISR1 > This function is synchronous. > This function is non-reentrant.

Table 4-44 CallbackFunc ASR 4.4

4.5.3 CallbackFunc ASR R19-11

Prototype	
`void **<CallbackFunc>** ( const Crypto_JobType *job, Crypto_ResultType result )`	
Parameter	
`job`	Points to the completed job's information structure. It contains a callbackID to identify which job is finished.
`result`	Contains the result of the cryptographic operation.
Return code	
`void`	none
Functional Description	
Notifies the CSM about the completion of the request with the result of the cryptographic operation.	
Particularities and Limitations	
> TASK | ISR2 | ISR1 > This function is synchronous. > This function is non-reentrant.	

Table 4-45 CallbackFunc ASR R19-11

4.5.4 CallbackFunc ASR R20-11

Prototype	
`void **<CallbackFunc>** (uint32 jobId, Crypto_ResultType result )`	
Parameter	
`jobId`	Identifies the job which is finished and caused the callback.
`result`	Contains the result of the cryptographic operation.
Return code	
`void`	none
Functional Description	
Notifies the CSM about the completion of the request with the result of the cryptographic operation.	

Particularities and Limitations
> TASK | ISR2 | ISR1 > This function is synchronous. > This function is non-reentrant.

Table 4-46 CallbackFunc ASR R20-11

4.6 Service Ports

4.6.1 Client Server Interface

A client server interface is related to a Provide Port at the server side and a Require Port at client side.

4.6.1.1 Provide Ports on CSM Side

At the Provide Ports of the CSM the API functions described in 4.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

5 Configuration

5.1 Overview

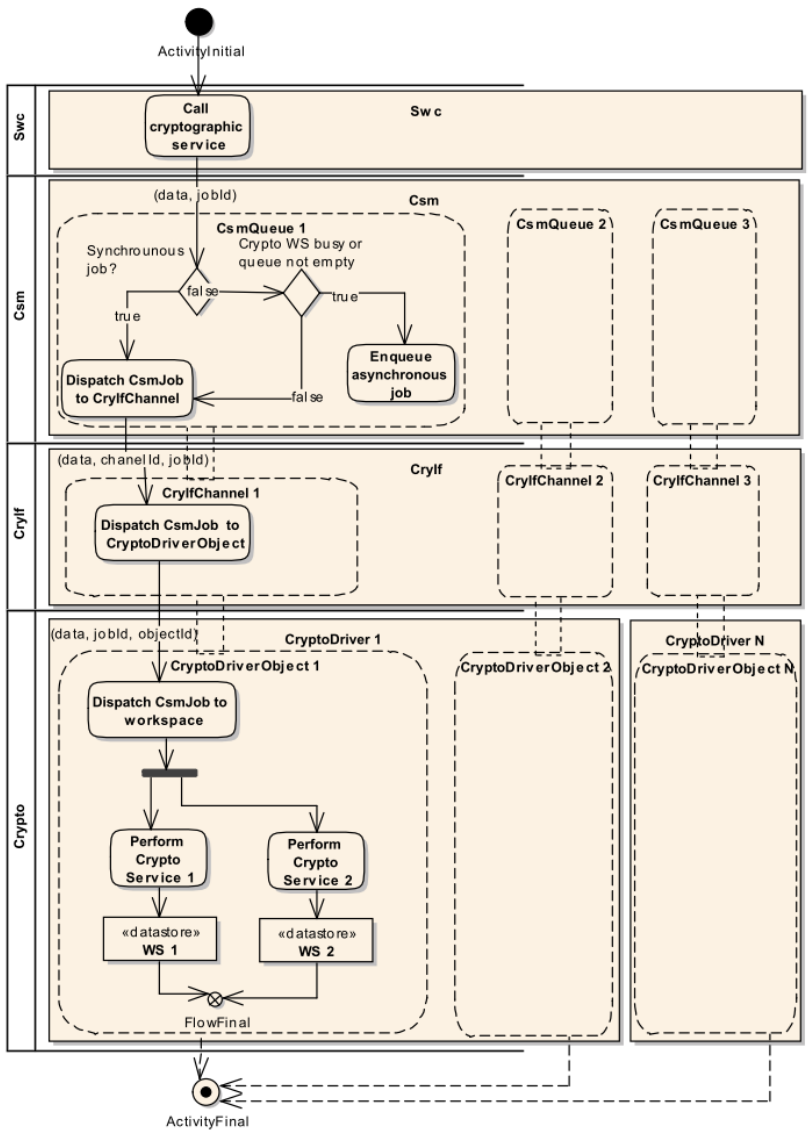


Figure 5-1 Structural overview and basic workflow

Figure 5-1 gives a structural overview and shows the basic workflow of the crypto stack. The Swc requests a cryptographic service of the CSM providing the necessary data and the id of the CsmJob which will be performed in the underlying CryptoDriverObject. The CsmJob contains all computational parameters for performing the cryptographic operation (see chapter 5.2).

At configuration time, each CsmJob is assigned to a CsmQueue, regardless of processing type (synchronous or asynchronous) of the job. Each CsmQueue is mapped to CryIfQueue and a CryIfQueue itself is mapped to a CryptoDriverObject of a Crypto Driver. At the crypto layer, N Crypto Drivers may coexist and do not interfere each other.

The CryptoDriverObject must allocate the required workspace and support the cryptographic operational in general. A job in computation occupies the cryptographic workspace of the cryptographic service; therefore jobs running in the same Crypto Driver object are processed in serialized fashion, whereas jobs running in different Crypto Driver objects or jobs of different cryptographic type may run in parallel.

All CSM attributes can be configured with the following tools:

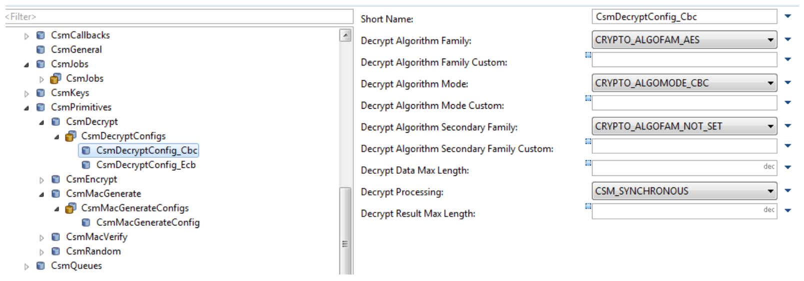
> Configuration in DaVinci Configurator 5 Pro

5.2 Configuration of a cryptographic operation

A cryptographic algorithm is mostly configurable and must be parametrized to operate in specific mode. A CsmJob encapsulates all these information and can be considered as an instance of a cryptographic algorithm realized in a CryptoDriverObject. The cryptographic operation and the operational mode are configured in a CsmPrimitive and is referred by a concrete CsmJob. A CsmPrimitive can be of one of following primitive service types:

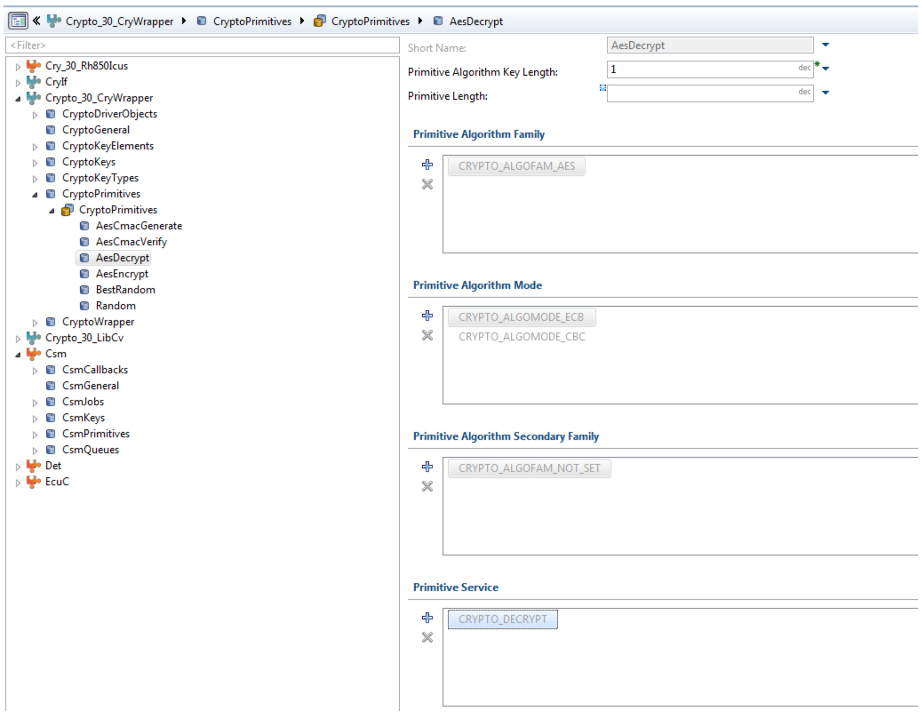
- MacGenerate
- MacVerifiy
- SignatureGenerate
- SignatureVerify
- Encrypt
- Decrypt
- Hash
- Random Generate
- AEADEncrypt
- AEADDecrypt
- KeySetValid
- KeySetInvalid
- RandomSeed
- KeyGenerate
- KeyDerive
- KeyExchangeCalcPubVal
- KeyExchangeCalcSecret

Further, each CsmPrimitive defines the algorithm family, mode and secondary family. The user has to ensure that the underlying destination Crypto Driver object supports the configured combination of these three parameters. The CryptoDriverObject itself must refer at least one CryptoPrimitive which matches in primitive service type, family, mode and secondary family (compare Figure 5-2, Figure 5-3 and Figure 5-4).



Short Name:	CsmDecryptConfig_Cbc
Decrypt Algorithm Family:	CRYPTO_ALGOFAM_AES
Decrypt Algorithm Family Custom:	
Decrypt Algorithm Mode:	CRYPTO_ALGOMODE_CBC
Decrypt Algorithm Mode Custom:	
Decrypt Algorithm Secondary Family:	CRYPTO_ALGOFAM_NOT_SET
Decrypt Algorithm Secondary Family Custom:	
Decrypt Data Max Length:	dec
Decrypt Processing:	CSM_SYNCHRONOUS
Decrypt Result Max Length:	dec

Figure 5-2 Configuration CsmPrimitive



Crypto_30_CryWrapper ▸ CryptoPrimitives ▸ CryptoPrimitives ▸ AesDecrypt

<Filter>

- Cry_30_Rh850Icus
- CryIf
- Crypto_30_CryWrapper
  - CryptoDriverObjects
  - CryptoGeneral
  - CryptoKeyElements
  - CryptoKeys
  - CryptoKeyTypes
  - CryptoPrimitives
    - CryptoPrimitives
      - AesCmacGenerate
      - AesCmacVerify
      - AesDecrypt
      - AesEncrypt
      - BestRandom
      - Random
  - CryptoWrapper
- Crypto_30_LibCv
- Csm
  - CsmCallbacks
  - CsmGeneral
  - CsmJobs
  - CsmKeys
  - CsmPrimitives
  - CsmQueues
- Det
- EcuC

Short Name: AesDecrypt

Primitive Algorithm Key Length: 1

Primitive Length:

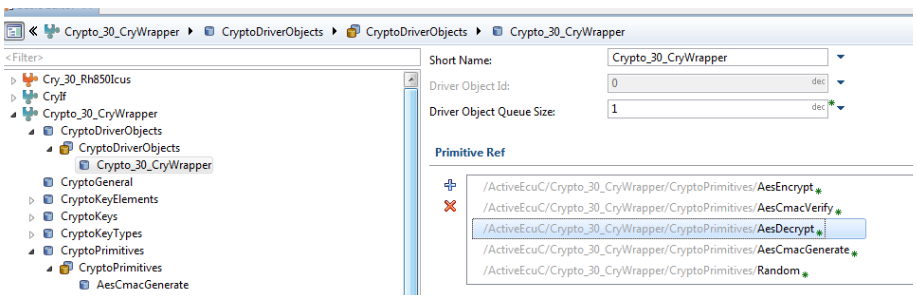
Primitive Algorithm Family: CRYPTO_ALGOFAM_AES

Primitive Algorithm Mode: CRYPTO_ALGOMODE_ECB, CRYPTO_ALGOMODE_CBC

Primitive Algorithm Secondary Family: CRYPTO_ALGOFAM_NOT_SET

Primitive Service: CRYPTO_DECRYPT

Figure 5-3 Configuration CryptoPrimitive



Crypto_30_CryWrapper ▸ CryptoDriverObjects ▸ CryptoDriverObjects ▸ Crypto_30_CryWrapper

<Filter>

- Cry_30_Rh850Icus
- CryIf
- Crypto_30_CryWrapper
  - CryptoDriverObjects
    - CryptoDriverObjects
      - Crypto_30_CryWrapper
  - CryptoGeneral
  - CryptoKeyElements
  - CryptoKeys
  - CryptoKeyTypes
  - CryptoPrimitives
    - CryptoPrimitives
      - AesCmacGenerate

Short Name: Crypto_30_CryWrapper

Driver Object Id: 0

Driver Object Queue Size: 1

Primitive Ref:

/ActiveEcuC/Crypto_30_CryWrapper/CryptoPrimitives/AesEncrypt
/ActiveEcuC/Crypto_30_CryWrapper/CryptoPrimitives/AesCmacVerify
/ActiveEcuC/Crypto_30_CryWrapper/CryptoPrimitives/AesDecrypt
/ActiveEcuC/Crypto_30_CryWrapper/CryptoPrimitives/AesCmacGenerate
/ActiveEcuC/Crypto_30_CryWrapper/CryptoPrimitives/Random

Figure 5-4 Configuration CryptoDriverObject

5.3 Configuration Variants

The CSM supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the CSM parameters depend on the supported configuration variants. For their definitions please see the Csm_bswmd.arxml file.

6 Cybersecurity

This chapter describes relevant information for a secure integration and configuration, so-called Cybersecurity Manual Instructions (CMI), of this component to fulfil identified Technical Cybersecurity Requirements (TCR). Additionally, functional security dependencies to other components are described.

6.1 Configuration

6.1.1 CMI-CSM-SoftwareComponentInterfaceAccess

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial, TCR-MSRC-AccessCertificates, TCR-MSRC-ProvideCryptoPrimitives, TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall configure all SW components that use CSM to only access CSM Jobs that were configured for the respective SW component.

Rationale:

Since there are no security checks built into the CSM interface, the using components or applications must be configured to not interfere with each other’s CSM Jobs. Misconfiguration may lead to security vulnerabilities of the system.

6.1.2 CMI-CSM-PseudoRandomNumberGeneratorSeed

Technical Cybersecurity Requirements: TCR-MSRC-ProvideCryptoPrimitives

The user of MICROSAR Classic shall use a true random number to seed the Pseudo Random Number Generator (RNG).

Rationale:

Especially when the RNG is used to create new keys, it must be seeded with a true random number, otherwise the created keys can be predicted. It makes sense to have a new seed or to add addition entropy at least on each startup of the system to avoid predictable start-up scenarios.

6.2 Runtime Interfaces: Application

6.2.1 CMI-CSM-PreAndPostJobCallouts

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial, TCR-MSRC-AccessCertificates, TCR-MSRC-ProvideCryptoPrimitives, TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall ensure when using pre- and post-job callouts, that the application that gets the callout does not alter the job object in any non-intended way.

Rationale:

Both, the pre- and post-job callouts have a pointer to the corresponding job. It is possible to manipulate the job object (e.g., adapt buffers, adjust lengths, and so on). Beware that the Csm is not able to check later whether the buffers are set up accordingly. More details about the pre- and post-job callouts can be found in chapter 2.5.

6.2.2 CMI-CSM-KeyWriteAccess

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial, TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall restrict write access to keys as much as possible to ensure that existing cryptographic material cannot be overwritten with Csm_KeyElementSet.

Rationale:

Csm_KeyElementSet could be used by an attacker to overwrite a valid key or certificate or to inject compromised cryptographic material.

6.2.3 CMI-CSM-KeyValidity

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial, TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall restrict access to KeySetValid as much as possible.

Rationale:

Csm_KeySetValid could be used by an attacker to set a stored key or certificate valid which was previously not considered valid.

6.2.4 CMI-CSM-KeyReadAccess

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial, TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall restrict read access to keys as much as possible to ensure that only the component that originally stored the key can retrieve it with Csm_KeyElementGet.

Rationale:

Csm_KeyElementGet could be used by an attacker to retrieve confidential key material.

6.2.5 CMI-CSM-KeyCopyAccess

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial, TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall restrict read and write access to keys as much as possible to ensure that Csm_KeyCopy, Csm_KeyElementCopy or Csm_KeyElementCopyPartial cannot overwrite existing keys and can only be used by the component that originally stored the key that shall be copied.

Rationale:

The above-mentioned interfaces could be used by an attacker to invalidate an important key used by another component or to overwrite an existing key with a less secure or known one.

6.2.6 CMI-CSM-KeyDecryption

Technical Cybersecurity Requirements: TCR-MSRC-ProvideCryptoPrimitives

The user of MICROSAR Classic shall restrict access to Csm_Decrypt and Csm_AEADDecrypt as much as possible.

Rationale:

An attacker could use one of the above-mentioned APIs to gain access to encrypted data on the system.

6.2.7 CMI-CSM-MacGeneration

Technical Cybersecurity Requirements: TCR-MSRC-ProvideCryptoPrimitives

The user of MICROSAR Classic shall restrict access to Csm_MacGenerate as much as possible.

Rationale:

An attacker with access to Csm_MacGenerate could send manipulated messages with a valid MAC.

6.2.8 CMI-CSM-SignatureGeneration

Technical Cybersecurity Requirements: TCR-MSRC-ProvideCryptoPrimitives, TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall restrict access to Csm_SignatureGenerate as much as possible.

Rationale:

An attacker with access to Csm_SignatureGenerate could generate a valid signature and impersonate a valid entity.

6.3 Runtime Interfaces: BSW

TCR	Depends on Component(s)	Comment
TCR-MSRC-AccessCryptoMaterial	CryIf	
TCR-MSRC-AccessCertificates	CryIf	
TCR-MSRC-ProvideCryptoPrimitives	CryIf	
TCR-MSRC-HandleCertificates	CryIf	

Table 6-1 Cybersecurity-relevant Dependencies for Runtime Interfaces

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
CSM	Crypto Service Manager
CryIf	Crypto Interface
Crypto	Crypto Driver

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CMI	Cybersecurity Manual Instruction
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PPORT	Provide Port
RPORT	Require Port
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
TCR	Technical Cybersecurity Requirement

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

> News
> Products
> Demo software
> Support
> Training data
> Addresses

www.vector.com