VECTOR

MICROSAR Secure Onboard Communication

Technical Reference

Version 9.1.0

Authors	vishho, visbms, visms, visgut, visbbk, viscpe
Status	Released

Document Information

History

Author	Date	Version	Remarks
vishho	2014-10-02	1.00.00	ESCAN00078719: AR4-667: CONC_607_SecureOnboardCommunication
vishho	2015-07-20	1.01.00	ESCAN00084099: FEAT-1475: SecOC-Extensions, TP, CSM and encryption
visms	2016-02-25	1.02.00	ESCAN00088549: FEAT-1631: Trigger Transmit API with SduLength In/Out according to ASR4.2.2
vishho	2016-11-11	1.03.00	FEATC-379: SecOC Release
visms	2017-02-27	2.00.00	FEATC-852: FEAT-2365: Support standalone distribution of AR4.3 SecOC
vishho	2017-03-13	2.01.00	ESCAN00094184: BETA version - the BSW module is in BETA state
vishho	2017-07-26	3.00.00	STORYC-919: Development Mode
vishho	2017-12-01	4.00.00	STORYC-2139: Special CAN FD Tx Padding logic STORYC-2138: Rx Container length calculation and Padding remove
visgut	2018-02-05	4.01.00	STORYC-3381: Configuration of secured area within a PDU
visbbk	2018-08-14	4.02.00	STORYC-6158: ASIL Release [SEC] SecOC: Provide PDU payload within message authentication failed callout STORYC-6156: ASIL Release [SEC] SecOC: ASIL Release Development mode for empty KeySlot
vishho	2018-08-22	4.02.00	STORYC-6157: ASIL Release Customer-specific Secure PDU Processing Callout in SecOC STORYC-5436: ASIL Release Tp Upper Layer interface
visbbk	2018-10-15	5.00.00	STORYC-6552: Support SecOCSecuredRxPduVerification parameter according ASR4.3.1
vishho	2018-12-04	5.01.00	STORYC-6930: Extension of the VerifyStatusOverride-API during runtime to ignore the reception of the Crypto-PDU.
visgut	2018-12-10	5.01.00	STORYC-7038: SecOC Immediate Rx/Tx handling
vishho	2019-03-13	5.02.00	STORYC-7871: Deny Retry in SecOc

viscpe	2019-06-18	6.00.00	STORYC-6478: RfC 80038 - Option to send wrong Authentication Information
visbbk	2019-06-25	6.00.00	STORYC-6476: SecOC - SecOC_VerifyStatusOverride and Verifiation Stauts Callout API update
visbbk	2019-07-05	6.01.00	STORYC-8098: Make Retry Transmit configurable
vishho	2020-09-02	7.00.00	COM-1617: Create timeout in SecOC for calls to Csm COM-1233: Introduce Queuing support in SecOC
vishho	2020-12-07	8.00.00	COM-2173 [AR-3215] Change of interface for SecOC_VerificationStatusService
visalefarth	2022-03-01	9.00.00	Product name updated to MICROSAR Classic.
Vishho	2023-04-19	9.00.00	SecOC support of Multi-Partitions
Vishho	2023-07-12	9.01.00	COM-2176 [SEC][IDPS] IdsM: support reporting of security events by SecOC

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_SecureOnboardCommunication.pdf	V4.3.0
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	V4.3.0
[3]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0

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 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.00	> Authentication of interface PDUs > Verification of interface PDUs
2.00.00	> Support of lower layer Tp interface > Support of CSM 4.2
3.01.00	> Trigger Transmit API with SduLength as in/out parameter
4.00.00	> Generic Freshness value interface > Verification status override api
5.00.00	> QM release > Support of splitting a Secured Message into an Authentic and an Cryptographic PDU > Support of Autosar 4.3 CSM > Post Build Loadable and Post Build Selectable
6.00.00	> Support standalone distribution of AR4.3 SecOC
7.01.00	> Removed support of CAL and CSM 4.2 > Removed SecOC internal Freshness Counter Support > SafeBSW release
8.00.00	> Dynamic DLC support
8.01.00	> CAN FD Container PDU Padding
9.01.00	> Upper Layer Tp interface
9.02.00	> Custom Verify/Generate callouts
9.03.00	> Support default authentication pattern for development mode
13.02.00	> CSM Job Timeout
13.03.00	> Reception queue
14.00.00	> RTE versification status service
16.00.00	> Multi Partion support
16.01.00	> IdsM Security Event support

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module SecOC as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	SecOC_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	SecOC_MODULE_ID	150 decimal (according to ref. [3])

* For the precise AUTOSAR Release 3.x please see the release specific documentation.

The AUTOSAR SecOC module provides a mechanism to authenticate and verify I-PDUs.
The SecOC module aims for resource-efficient and practicable authentication mechanisms for critical data on the level of PDUs.

2.1 Architecture Overview

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

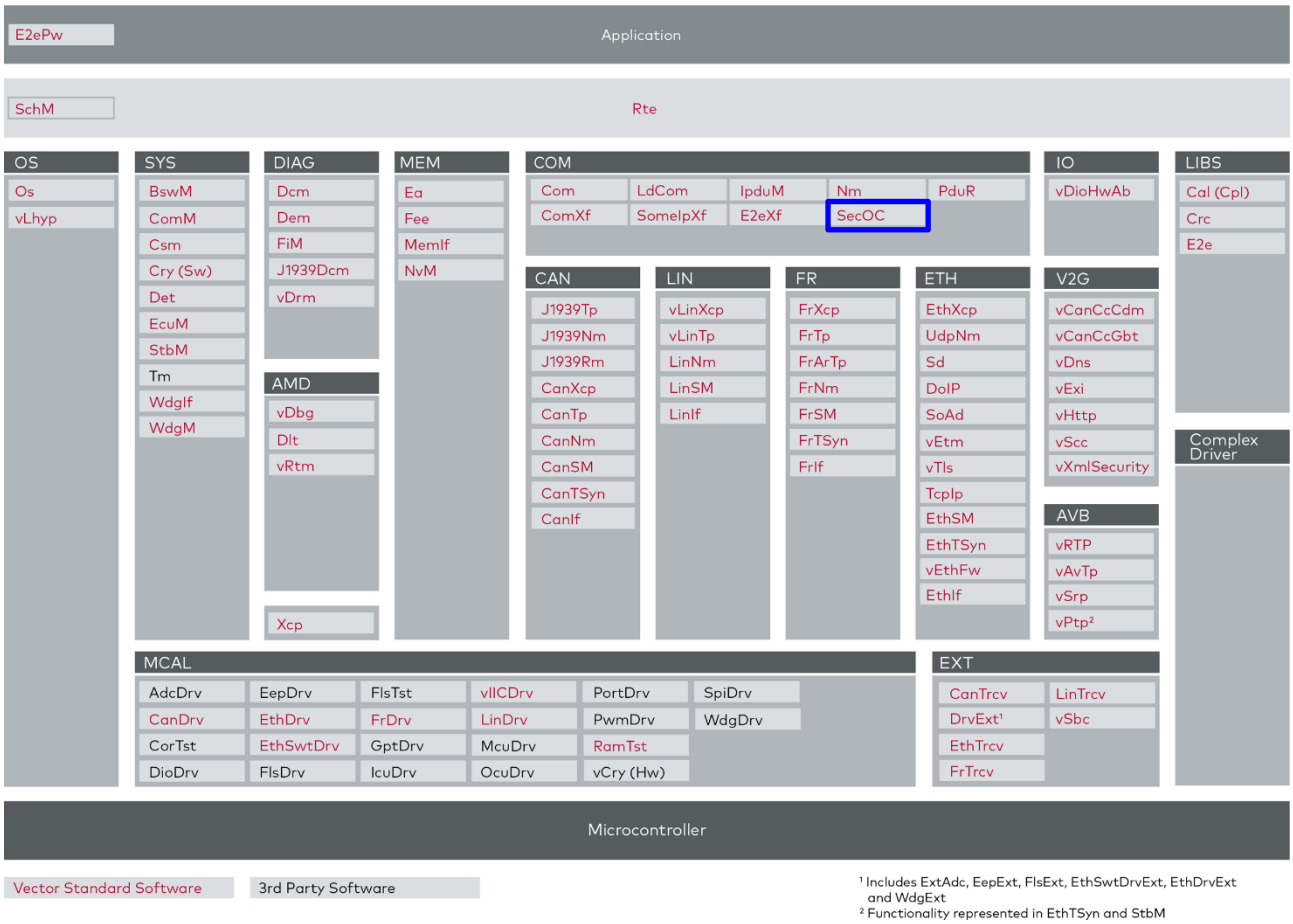

Figure 2-1 AUTOSAR 4.2 Architecture Overview

3 Functional Description

3.1 Features

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

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

- Table 3-1 Supported AUTOSAR standard conform features
- Table 3-2 Not supported AUTOSAR standard conform features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Authentication during direct transmission
Authentication during triggered transmission
Verification during bus interface reception
Authentication during transport protocol transmission
Verification during transport protocol reception
Use of CSM 4.3
RTE Service Interface: > Verification Status Service > Counter Management Service > Generic Freshness Value interface > Verification Status Configuration Service
Support of splitting a Secured Message into an Authentic and an Cryptographic PDU

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
PduR APIs according to Autosar 4.3
Reception Overflow Strategie Replace

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Secured PDUs

A Secured PDU is the combination of:

Authentic PDU | Truncated Freshness Value | Truncated Authenticator (Big Endian Style)

To generate the Authenticator following data is used:

Data Id | Authentic PDU (partly or complete) | Complete Freshness value

3.1.2.1 Reception of Secured PDUs

The SecOC is used to verify received Secured PDUs. If the verification is successful the Authentic PDU is passed to the upper layer.

The SecOC will forward the received PDU without verification if secured rx PDU verification is disabled.

3.1.2.2 Transmission of Secured PDUs

The SecOC adds to an Authentic PDU the truncated Freshness Value and the Truncated Authenticator and passes the build Secured PDU to the Lower Layer.

If the lower layer reports a failed transmission the SecOC will retry the transmission the next main function cycle if the retry failed transmission request feature is enabled.

If the feature is disabled the SecOC will discard the transmission and call a negative confirmation in case of TP.

3.1.2.3 Secured PDU Collections

A Secured PDU Collection splits a Secured PDU into two separate PDUs, the Authentic PDU and the Cryptographic PDU. The Authentic PDU contatins the Payload passed from the Upper Layer and the Cryptographic PDU contains the truncated freshness value and the truncated authenticator.

To improve the reliability, the two messages can be linked by a Message Linker. The message linker is a byte from the Authentic PDU that is attached to the Cryptographic PDU. On Reception side the verification is only started if the message linker of the authentic and the cryptographic PDU are equal.

No verification will be performed if the secured rx PDU verification is disabled.

3.1.2.4 Secured Area

In general, the SecOC uses the complete Authentic PDU to calculate the authenticator. However, it is also possible to configure the secured content as a portion of the Authentic PDU. In this case, only a configurable number of bytes starting from a given offset are considered for the authenticator. Nonetheless, the complete Data Id and the complete Freshness value are always used for MAC generation or verification.

3.1.3 Generic Freshness value interface

The Generic Freshness value interface is used by the SecOC to get a Freshness value to verify or generate a MAC.

On Reception side the SecOC can use two different generic freshness value interfaces to get the freshness verify value. Which callout is used by the SecOC can be configured per Rx Processing.

SecOC_GetRxFreshnessAuthData this callout is used if the Freshness Management Component needs additional information from the Authentic PDU to generate the freshness verify value. (For details see 5.4.2.6)

SecOC_GetRxFreshness this callout can be used for all common freshness value use cases, e.g. Freshness Counter and Freshness Timestamp. (For details see 5.4.2.7)

On Transmission side the SecOC can use two different generic freshness value interfaces to get the freshness verify value. Which callout is used by the SecOC can be configured per Tx Processing.

SecOC_GetTxFreshnessTruncData this callout is used if the Freshness Management Component provides the complete freshness value and the truncated freshness value. (For details see 5.4.2.3)

SecOC_GetTxFreshness this callout is used if the Freshness Management Component provides only the complete freshness value. (For details see 5.4.2.4)

After a transmit request is passed successful to the lower layer component the callout SecOC_SPduTxConfirmation is called, this callout can be used to increment the freshness value within the Freshness Management Component. (For details see 5.4.2.5)

3.2 Development Mode

If the development mode is active during runtime all PDUs will be passed to the Upper Layer, even if the verification fails.

The development mode must be enabled in the configuration with the configuration switch /MICROSAR/SecOC/SecOCGeneral/SecOCEnableDevelopmentMode.

To enable the development mode during runtime the API SecOC_SetDevelopmentMode must be called.

A default authentication pattern can be configured which will be used as freshness and authentication value in case the authentication fails and the development mode is enabled.

A transmit to the lower layer will still be performed with the default authentication pattern information.

When no pattern is defined but the development mode is enabled, the default authentication pattern is set to zero. This value will be used as freshness and authentication value. A transmit to the lower layer will still be performed.

3.3 Send Default Authentication Information

This feature provides the option of sending unauthenticated PDUs to the lower layer.

This Feature must be enabled in the configuration by configuring the parameter /MICROSAR/SecOC/SecOCGeneral/SecOCDefaultAuthenticationInformationPattern.

Additionaly it must be enabled for specific specifice Freshness Value IDs by calling the API SecOC_SendDefaultAuthenticationInformation during runtime.

If a MAC can not be provided durning transmission and this feature is enabled for the given Freshness Value ID of the PDU, the transmission is still performed.

In this case the SecOCDefaultAuthenticationInformationPattern is used for all bytes of the Freshness Value and the Authenticator.

3.4 CAN FD Padding Support for Container PDUs

This feature adds padding bytes after the MAC generation of a container PDU to prevent that the CAN driver adds padding bytes. On reception side, the container headers are used to calculate the actual length of the PDU and the padding bytes are removed before the MAC verification.

Figure 3-1 CAN FD Container PDU with padding

Note
To use this feature a license option is required.

3.5 Customer specific Secured PDU processing

The SecOC provides the option to handle the verification and generation of the MAC outside the SecOC.

If this feature is enabled the freshness value construction and “Data To Authenticate” construction is handled within callouts.

The SecOC will use the callout SecOC_VerifyAuthenticationInfo to get the verification result on Rx Side and on Tx Side the SecOC will use the callout SecOC_GenerateAuthenticationInfo to generate the truncated MAC.

3.6 Upper Layer Tp Interface

The SecOC provides the option to forward a received authentic PDU to the upper layer by the usage of the upper layers Tp interface and the SecOC provides the option to get the data of a transmitted authentic PDU by the usage of the upper layer Tp interface.

On reception side the StartOfReception will directly be forwarded to the upper layer. If the verification is successful the data will be forwarded to the upper layer. A successful verification will be finished with a call of TpRxIndicatoin with a positive result, a failed verification will be indicated to the upper layer with a call of TpRxIndicatoin with a negative result.

On transmission side the SecOC will get the data right before the generation of the MAC within the main function. The lower layer Tp Confirmation will be forwarded to the upper layer Tp. In case the generation of the MAC failed the upper layer will get a negative TpTxConfirmation.

Note
The Tp upper layer interface can only be used if the PDU is received/transmitted via a lower layer Tp.

3.7 Initialization

If not already done by the startup code, the statically initialized RAM variables

must be initialized by calling SecOC_InitMemory().

The SecOC itself is initialized by calling SecOC_Init().The module can be reset with

SecOC_DeInit().

3.8 Pdu Processing

Authentication and verification can be either be performed In the task context of the respective main function (deferred processing) or in the call context of the requested service (immediate processing).

3.8.1 Deferred Processing

The SecOC has the main functions SecOC_MainFunctionRx() and SecOC_MainFunctionTx(). By default, authentication or verification of a PDU will always take place within the main functions.

3.8.2 Immediate Processing

To reduce the processing latency, SecOCPduProcessing can be set to IMMEDIATE. In this case authentication or verification of a PDU will take place in the call context of the respective service. However, immediate processing is only permitted, if all the following conditions are satisfied:

- SecOCPduType is set to SECOC_IFPDU
- Dest Pdu Data Provision of the respective routing path does not involve TRIGGER_TRANSMIT
- The referred CsmJob is used exclusively

Further, in case if the adjacent CSM module performs requests asynchronously, it's optionally configurable, if the result, provided to configured callback function, shall be processed immediately. However, to avoid resursive calling loops on Rx side, immediate processing is only initiated in case of positive verification result. To enable this option, SecOCImmediateProcessingOfCsmCallback has to be set to TRUE.

Remark: Any configured retrys are still performed in the respective main functions.

3.9 CSM asynchronous processing timeout

For SecOCRxPduProcessing or SecOCTxPduProcessing an asynchronous CSM Job timeout can be configured with the parameter SecOCAsyncCSMJobTimeout. The timeout monitoring starts when a Job in the CSM is started and the timeout monitoring is stopped when the CSM indicates that the asynchronous job is finished by a callout call.

When a timeout occurs the SecOC cancels the CSM Job and frees the buffer so that new PDUs can be received again.

3.10 Reception queue

If the SecOCReceptionOverflowStrategy is set to QUEUE and the SecOCReceptionQueueSize is configured for a SecOCRxPduProcessing the SecOC queues the received Secured PDU, if the internal processing buffer is blocked by a currently processed Secured PDU.

The queue is processed as a FIFO queue.

3.11 Multipartition

In order to provide a load distribution amongst different partitions (cores), the main threads of execution in the SecOC module, namely the respective MainFunctions are assigned to different partitions. This way the flow of reception/transmission stays within the scope of a single partition.

3.11.1 General configuration

The general concept for the multipartition support of the SecOC is, that the SecOC(Rx/Tx)PduProcessing are processed in the MainFunction which is referenced via the SecOC(Rx/Tx)PduMainFunctionRef. The Main Functions are assigned to a partition via the SecOCMainFunction(Rx/Tx)/SecOCMainFunction(Rx/Tx)PartitionRef.

3.11.2 Initialization and Deinitialization

The SecOC shares all RAM between partitions, due to that the SecOC_Init() and SecOC_DeInit() functions must only be called once to initialize/deinitialize the SecOC variables.

3.11.3 Restrictions

The following restrictions apply for the SecOC Multipartition feature:

- One Partition can only have one SecOCMainFunctionRx and one SecOCMainFunctionTx
- SecOC(Rx/Tx)PduProcessing with the same SecOCFreshnessValueId must belong to the same Partition/MainFunction.

3.11.4 Security Events

The SecOC can report Security Events to the IdsM. The Security Events are configured in the container: SecOC/SecOCGeneral/SecOCSecurityEventRefs.

Following Security Events are supported:

Name	Description	Context Data
SECOC_SEV_MAC_VERIFICATION_FAILED	MAC verification of a received PDU failed.	DataId (2 Byte)
SECOC_SEV_FRESHNESS_NOT_AVAILABLE	Failed to get freshness value from FvM.	None.

3.12 Error Handling

3.12.1 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 `SecOC_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 SecOC ID is 150.

The reported service IDs identify the services which are described in 5.1. The following table presents the service IDs and the related services:

Service ID	Service
0x01	SECOC_SID_INIT
0x02	SECOC_SID_GET_VERSION_INFO
0x03	SECOC_SID_TRANSMIT
0x04	SECOC_SID_CANCEL_TRANSMIT
0x05	SECOC_SID_DE_INIT
0x08	SECOC_SID_FRESHNESS_VALUE_READ
0x09	SECOC_SID_FRESHNESS_VALUE_WRITE
0x0B	SECOC_SID_VERIFY_STATUS_OVERRIDE
0x0C	SECOC_SID_RX_INDICATION
0x0E	SECOC_SID_TX_CONFIRMATION
0x0F	SECOC_SID_TRIGGER_TRANSMIT
0x14	SECOC_SID_MAIN_FUNCTION_RX
0x15	SECOC_SID_MAIN_FUNCTION_TX
0x43	SECOC_SID_COPY_TX_DATA
0x44	SECOC_SID_COPY_RX_DATA
0x45	SECOC_SID_TP_RX_INDICATION
0x46	SECOC_SID_START_OF_RECEPTION
0x48	SECOC_SID_TP_TX_CONFIRMATION
0x50	SECOC_SID_SET_DEVELOPMENT_MODE
0x51	SECOC_SID_SEND_DEFAULT_AUTHENTICATION_INFORMATION
0x80	SECOC_SID_INTERNAL

Table 3-3 Service IDs

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

Error Code	Description
0x01	An API service was called with a NULL pointer
0x02	API service used without module initialization or PduR_Init called

Error Code	Description
0x03	Invalid I-PDU identifier
0x04	Crypto service failed
0x05	Unable to get Freshness Value
0x06	Freshness Value at limit
0x07	An API service was called with an invalid parameter

Table 3-4 Errors reported to DET

4 Integration

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

4.1 Scope of Delivery

The delivery of the SecOC contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
SecOC.c	Source file of the SecOC module
SecOC.h	Main header file which shall be included by modules using the SecOC

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool [config tool].

File Name	Description
SecOC_Cfg.h	This file contains: > global constant macros > global function macros > global data types and structures > global data prototypes > global function prototypes of CONFIG-CLASS PRE-COMPILE data.
SecOC_Cbk.h	This is the generated header file of SecOC containing prototypes for lower layers.
SecOC_Cot.h	This is the generated header file of SecOC containing callout prototypes.
SecOC_Lcfg.h	This file contains: > global constant macros > global function macros > global data types and structures > global data prototypes > global function prototypes of CONFIG-CLASS LINK data.
SecOC_Lcfg.c	This file contains: > local constant macros > local function macros > local data types and structures

File Name	Description
	> local data prototypes > local data > global data of CONFIG-CLASS LINK data.
SecOC_PBcfg.h	This file contains: > global constant macros > global function macros > global data types and structures > global data prototypes > global function prototypes of CONFIG-CLASS POST-BUILD data.
SecOC_PBcfg.c	This file contains: > local constant macros > local function macros > local data types and structures > local data prototypes > local data > global data of CONFIG-CLASS POST-BUILD data.
SecOC_Types.h	This file contains the static type definitions.
SecOC_Lcfg_<PartitionName>.h	This file contains: > partition specifc function prototypes
SecOC_Lcfg_<PartitionName>.c	This file contains: > partition specifc functions

Table 4-2 Generated files

4.2 Critical Sections

The handling of entering and leaving critical sections is provided by the RTE.

Following critical sections are offered:

- SECOC_EXCLUSIVE_AREA_TXSTATE

  Used to provide exclusive access to the Tx state value.

  Used in following functions:

  - `SecOC_MainFunctionTx`
  - `SecOC_Transmit`
  - `SecOC_TxConfirmation`
  - `SecOC_TriggerTransmit`
- SECOC_EXCLUSIVE_AREA_RXSTATE

  Used to provide exclusive access to the Rx state value.

Used in following functions:

- `SecOC_MainFunctionRx`
- `SecOC_RxIndication`

5 API Description

5.1 Services provided by SecOC

5.1.1 SecOC_InitMemory

Prototype	
`void SecOC_InitMemory (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Function for *_INIT_*-variable initialization.	
Particularities and Limitations	
> Module must not be initialized. Function shall be called from task level Service to initialize module global variables at power up. This function can be used to initialize the variables in *_INIT_* sections in case they are not initialized by the startup code.	
Call context	
FunctionCallcontext	

Table 5-1 SecOC_InitMemory

5.1.2 SecOC_Init

Prototype	
`void SecOC_Init (const SecOC_ConfigType *config)`	
Parameter	
config [in]	Configuration structure for initializing the module
Return code	
void	none
Functional Description	
Initialization function.	
Particularities and Limitations	
Specification of module initialization > Interrupts have to be disabled.The module has to be uninitialized. This function initializes the module SecOC. It initializes all variables and sets the module state to initialized.	
Call context	
FunctionCallcontext	

Table 5-2 SecOC_Init

5.1.3 SecOC_DeInit

Prototype	
`void SecOC_DeInit (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Deinitialization function.	
Particularities and Limitations	
Specification of module initialization > Interrupts have to be disabled. The module has to be initialized. This function stops the secure onboard communication. All buffered I-PDU are removed and have to be optained again, if needed, after SecOC_Init has been called. By a call to SecOC_DeInit the AUTOSAR SecOC module is put into an not initialized state (SecOC_UNINIT).	
Call context	
FunctionCallcontext	

Table 5-3 SecOC_DeInit

5.1.4 SecOC_GetVersionInfo

Prototype	
`void SecOC_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer to where to store the version information
Return code	
void	none
Functional Description	
Returns the version information.	
Particularities and Limitations	
> Input parameter must not be NULL.Function shall be called from task level SecOC_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component.	
Call context	
FunctionCallcontext	

Table 5-4 SecOC_GetVersionInfo

5.1.5 SecOC_Transmit

Prototype	
`Std_ReturnType` **`SecOC_Transmit`** `(PduIdType id, const PduInfoType *info)`	
Parameter	
id [in]	ID of the Authentic I-PDU to be transmitted.
info [in]	A pointer to a structure with Authentic I-PDU related data that shall be transmitted: data length and pointer to I-SDU buffer
Return code	
Std_ReturnType	Std_ReturnType E_OK: request is accepted by the SecOC module. transmission is continued. E_NOT_OK: request is not accepted by the SecOC module. transmission is aborted.
Functional Description	
Transmits an Authentic I-PDU.	
Particularities and Limitations	
Function is called by the PDUR to request authentication and transmission of an Authentic I-PDU.	
Call context	
FunctionCallcontext	

Table 5-5 SecOC_Transmit

5.1.6 SecOC_VerifyStatusOverride

Prototype	
`Std_ReturnType` **`SecOC_VerifyStatusOverride`** `(uint16 freshnessValueID, uint8 overrideStatus, uint8 numberOfMessagesToOverride)`	
Parameter	
freshnessValueID [in]	ID of the Freshness Value which when used to authenticate data, results in SecOC_VerifyStatus equal to OverrideStatus independent of the actual authentication status.
overrideStatus [in]	0 = Override VerifyStatus to "Fail" until further notice 1 = Override VerifyStatus to "Fail" until NumberOfMessagesToOverride is reached 2 = Cancel Override of VerifyStatus 40 = Override VerifyStatus to "Pass" until further notice. Only available if SecOCEnableForcedPassOverride is set to TRUE 41 = Override VerifyStatus to "Pass" until NumberOfMessagesToOverride is reached. Only available if SecOCEnableForcedPassOverride is set to TRUE 0x41 = Until NumberOfMessagesToOverride is reached, authenticator verification is not performed Only available if SecOCEnableForcedPassOverride is set to TRUE 0x43 = Authenticator verification is not performed until further notice. Only available if SecOCEnableForcedPassOverride is set to TRUE 0x40 = Override VerifyStatus to “PASS_UNTIL_NOTICE”, verification is performed and PDU sent to upper layer. Only available if SecOCEnableForcedPassOverride is set to TRUE 41. 0x42 = Override VerifyStatus to “PASS_UNTIL_LIMIT” until number of messages is reached. Only available if SecOCEnableForcedPassOverride is set to TRUE
numberOfMessagesToOverride	Number of sequential VerifyStatus to override when using a specific

[in]	counter for authentication verification. This is only considered when OverrideStatus is equal to 1 or 41.
Return code	
Std_ReturnType	Std_ReturnType E_OK: request successful E_NOT_OK: request failed
Functional Description	
Sets verification status override.	
Particularities and Limitations	
This service provides the ability to override the VerifyStatus with "Fail" or "Pass" when using a specific Freshness Value to verify authenticity of data making up an I-PDU. Using this interface, VerifyStatus may be overridden 1. Indefinitely for received I-PDUs which use the specific Freshness Value for authentication verification 2. For a number of sequentially received I-PDUs which use the specific Freshness Value for authentication verification.	
Call context	
> TASK	

Table 5-6 SecOC_VerifyStatusOverride

5.1.7 SecOC_SetDevelopmentMode

Prototype	
`void SecOC_SetDevelopmentMode (boolean enableDevMode)`	
Parameter	
enableDevMode [in]	TRUE - Development Mode is enabled. FALSE - Development Mode is disabled.
Return code	
void	None
Functional Description	
Enables/Disables the development mode.	
Particularities and Limitations	
-	
Call context	
> TASK	

Table 5-7 SecOC_SetDevelopmentMode

5.1.8 SecOC_SendDefaultAuthenticationInformation

Prototype	
`void SecOC_SendDefaultAuthenticationInformation (uint16 freshnessValueID, boolean sendDefaultAuthenticationInformation)`	
Parameter	
freshnessValueID [in]	ID of the Freshness Value for which sending SecOCDefaultAuthenticationInformationPattern should be enabled.
sendDefaultAuthenticationInformation	TRUE - sending SecOCDefaultAuthenticationInformationPattern

[in]	shall be enabled for given FreshnessValueID. FALSE – sending SecOCDefaultAuthenticationInformationPattern shall be disabled for given FreshnessValueID
Return code	
Std_ReturnType	Std_ReturnType E_OK: request successful. E_NOT_OK: request failed.
Functional Description	
Enables/Disables the transmission of unauthenticated PDUs for the given Freshness Value ID	
Particularities and Limitations	
-	
Call context	
> TASK	

5.1.9 SecOC_MainFunctionRx

Prototype	
`void SecOC_MainFunctionRx (void)`	
Parameter	
Void	None
Return code	
Void	None
Functional Description	
This function verifies the received Secured PDUs.	
Particularities and Limitations	
The function is called by the BSW Scheduler.	
Call Context	
This function must be called on task level.	

Table 5-8 SecOC_ MainFunctionRx

5.1.10 SecOC_MainFunctionTx

Prototype	
`void SecOC_MainFunctionTx (void)`	
Parameter	
Void	None
Return code	
Void	None
Functional Description	
Authenticates the Authentic PDUs before transmission.	

Particularities and Limitations
The function is called by the BSW Scheduler.
Call Context
This function must be called on task level.

Table 5-9 SecOC_ MainFunctionTx

5.2 Services used by SecOC

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

Component	API
DET	Det_ReportError
PDUR	PduR_SecOCTransmit
PDUR	PduR_SecOCRxIndication
PDUR	PduR_SecOCTxConfirmation
PDUR	PduR_SecOCTpStartOfReception
PDUR	PduR_SecOCTpCopyRxData
PDUR	PduR_SecOCTpRxIndication
PDUR	PduR_SecOCTpCopyTxData
PDUR	PduR_SecOCCancelTransmit
PDUR	PduR_SecOCTpTxConfirmation
CSM	Csm_MacGenerate
CSM	Csm_MacVerify
IdsM	IdsM_SetSecurityEvent
IdsM	IdsM_SetSecurityEventWithContextData

Table 5-10 Services used by the SecOC

5.3 Callback Functions

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

5.3.1 SecOC_RxIndication

Prototype	
`void SecOC_RxIndication (PduIdType id, const PduInfoType *info)`	
Parameter	
id [in]	ID of the received Secured I-PDU.
info [in]	A pointer to a structure with Secured I-PDU related data: data length and pointer to I-SDU buffer.
Return code	
void	none
Functional Description	
Callback is called when a Secured I-PDU is received.	
Particularities and Limitations	
Called by the PDUR to indicate direct reception of a Secured I-PDU from a lower layer communication interface. This call triggers the verification of the received Secured I-PDU.	
Call context	
FunctionCallcontext	

Table 5-11 SecOC_RxIndication

5.3.2 SecOC_TxConfirmation

Prototype	
`void SecOC_TxConfirmation (PduIdType id)`	
Parameter	
id [in]	ID of the transmitted Secured I-PDU.
Return code	
void	none
Functional Description	
Callback is called to confirm a transmission.	
Particularities and Limitations	
The lower layer communication interface module confirms the transmission of a Secured I-PDU via PDUR.	
Call context	
FunctionCallcontext	

Table 5-12 SecOC_TxConfirmation

5.3.3 SecOC_TriggerTransmit

Prototype	
`Std_ReturnType` **`SecOC_TriggerTransmit`** `(PduIdType id, PduInfoType *info)`	
Parameter	
id [in]	ID of the Authentic I-PDU to be transmitted.
info [in,out]	Contains a pointer to a buffer (SduDataPtr) where the SDU data shall be copied to, and the available buffer size in SduLengh. On return, the service will indicate the length of the copied SDU data in SduLength.
Return code	
Std_ReturnType	E_OK SDU has been copied and SduLength indicates the number of copied bytes.
Std_ReturnType	E_NOT_OK No data has been copied, because SecOC is not initialized or TxPDUId is not valid or PDUInfoPtr is NULL_PTR or SduDataPtr is NULL_PTR or SduLength is too small
Functional Description	
Callback is called ba the Lower Layer to get a Secured I-PDU.	
Particularities and Limitations	
The function is called when a lower layer communication module expects a Secured I-PDU to be transmitted. The SecOC module copies the Secured I-PDU into the buffer of the lower layer communication module.	
Call context	
FunctionCallcontext	

Table 5-13 SecOC_TriggerTransmit

5.3.4 SecOC_TpRxIndication

Prototype	
`void` **`SecOC_TpRxIndication`** `(PduIdType id, Std_ReturnType result)`	
Parameter	
id [in]	Identification of the received I-PDU.
result [in]	Result of the reception.
Return code	
void	none
Functional Description	
Callback is called to indicate a completed reception.	
Particularities and Limitations	
The function is called after an I-PDU has been received via the TP API, the result indicates whether the transmission was successful or not.	
Call context	
FunctionCallcontext	

Table 5-14 SecOC_TpRxIndication

5.3.5 SecOC_TpTxConfirmation

Prototype	
`void SecOC_TpTxConfirmation (PduIdType id, Std_ReturnType result)`	
Parameter	
id [in]	Identification of the transmitted I-PDU.
result [in]	Result of the transmission of the I-PDU.
Return code	
void	none
Functional Description	
Callback is called to indicate a completed transmission.	
Particularities and Limitations	
This function is called after the I-PDU has been transmitted on its network, the result indicates whether the transmission was successful or not.	
Call context	
FunctionCallcontext	

Table 5-15 SecOC_TpTxConfirmation

5.3.6 SecOC_CopyRxData

Prototype	
`BufReq_ReturnType SecOC_CopyRxData (PduIdType id, const PduInfoType *info, PduLengthType *bufferSizePtr)`	
Parameter	
id [in]	Identification of the received I-PDU.
info [in]	Provides the source buffer (SduDataPtr) and the number of bytes to be copied (SduLength). An SduLength of 0 can be used to query the current amount of available buffer in the upper layer module. In this case, the SduDataPtr may be a NULL_PTR.
bufferSizePtr [out]	Available receive buffer after data has been copied.
Return code	
BufReq_ReturnType	BufReq_ReturnType BUFREQ_OK: Data copied successfully. BUFREQ_E_NOT_OK: Data was not copied because an error occurred.
Functional Description	
This callback copies the received I-PDU segment.	
Particularities and Limitations	
This function is called to provide the received data of an I-PDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the I-PDU data. The size of the remaining data is written to the	

position indicated by bufferSizePtr.
Call context
FunctionCallcontext

Table 5-16 SecOC_CopyRxData

5.3.7 SecOC_CopyTxData

Prototype	
`BufReq_ReturnType SecOC_CopyTxData (PduIdType id, const PduInfoType *info, RetryInfoType *retry, PduLengthType *availableDataPtr)`	
Parameter	
id [in]	Identification of the transmitted I-PDU.
info [in]	Provides the destination buffer (SduDataPtr) and the number of bytes to be copied (SduLength). If not enough transmit data is available, no data is copied by the upper layer module and BUFREQ_E_BUSY is returned. The lower layer module may retry the call. An SduLength of 0 can be used to indicate state changes in the retry parameter or to query the current amount of available data in the upper layer module. In this case, the SduDataPtr may be a NULL_PTR.
retry [in]	This parameter is used to acknowledge transmitted data or to retransmit data after transmission problems. If the retry parameter is a NULL_PTR, it indicates that the transmit data can be removed from the buffer immediately after it has been copied. Otherwise, the retry parameter must point to a valid RetryInfoType element. If TpDataState indicates TP_CONFPENDING, the previously copied data must remain in the TP buffer to be available for error recovery. TP_DATACONF indicates that all data that has been copied before this call is confirmed and can be removed from the TP buffer. Data copied by this API call is excluded and will be confirmed later. TP_DATARETRY this parameter will lead to the negative return of BUFREQ_E_NOT_OK.
availableDataPtr [out]	Indicates the remaining number of bytes that are available in the upper layer module's Tx buffer. availableDataPtr can be used by TP modules that support dynamic payload lengths (e.g. FrIsoTp) to determine the size of the following CFs.
Return code	
BufReq_ReturnType	BufReq_ReturnType BUFREQ_OK: Data has been copied to the transmit buffer completely as requested. BUFREQ_E_BUSY: Request could not be fulfilled, because the required amount of Tx data is not available. The lower layer module may retry this call later on. No data has been copied. BUFREQ_E_NOT_OK: Data has not been copied. Request failed.
Functional Description	
This callback copies the transmitted I-PDU segment.	
Particularities and Limitations	
This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the I-PDU data unless retry->TpDataState is TP_DATARETRY. In this case the	

function restarts to copy the data beginning at the offset from the current position indicated by retry->TxTpDataCnt. The size of the remaining data is written to the position indicated by availableDataPtr.
Call context
FunctionCallcontext

Table 5-17 SecOC_CopyTxData

5.3.8 SecOC_StartOfReception

Prototype	
`BufReq_ReturnType` **`SecOC_StartOfReception`** `(PduIdType id, const PduInfoType *info, PduLengthType TpSduLength, PduLengthType *bufferSizePtr)`	
Parameter	
id [in]	Identification of the I-PDU.
info [in]	Pointer to a PduInfoType structure containing the payload data (without protocol information) and payload length of the first frame or single frame of a transport protocol I-PDU reception. Depending on the global parameter MetaDataLength, additional bytes containing MetaData (e.g. the CAN ID) are appended after the payload data, increasing the length accordingly. If neither first/single frame data nor MetaData are available, this parameter is set to NULL_PTR.
TpSduLength [in]	Total length of the N-SDU to be received.
bufferSizePtr [out]	Available receive buffer in the receiving module. This parameter will be used to compute the Block Size (BS) in the transport protocol module.
Return code	
BufReq_ReturnType	BufReq_ReturnType BUFREQ_OK: Connection has been accepted. bufferSizePtr indicates the available receive buffer; reception is continued. If no buffer of the requested size is available, a receive buffer size of 0 shall be indicated by bufferSizePtr. BUFREQ_E_NOT_OK: Connection has been rejected; reception is aborted. bufferSizePtr remains unchanged. BUFREQ_E_OVFL: No buffer of the required length can be provided; reception is aborted. bufferSizePtr remains unchanged.
Functional Description	
This callback is called to indicate the start of a segmented reception.	
Particularities and Limitations	
This function is called at the start of receiving an N-SDU. The N-SDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF).	
Call context	
FunctionCallcontext	

Table 5-18 SecOC_StartOfReception

5.4 Configurable Interfaces

5.4.1 Notifications

5.4.2 Callout Functions

At its configurable interfaces the SecOC defines callout functions. The declarations of the callout functions are provided by the BSW module, i.e. the SecOC. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The SecOC callout function declarations are described in the following tables:

5.4.2.1 SecOC_VerificationStatusCallout

Prototype	
`void` **`SecOC_VerificationStatusCallout`** `(SecOC_VerificationStatusType verificationStatus )`	
Parameter	
SecOC_VerificationStatusType	verificationStatus
Return code	
void	none
Functional Description	
Service is used to propagate the status of each verification attempt from the SecOC module to other modules. This service can be configured such that: > Only: “False” Verification Status is propagated to modules > Both: “True” and “False” Verification Status is propagated to modules > None: “No Verification Status is propagated“	
Particularities and Limitations	
> none	
Call context	
> task context	

Table 5-19 [SecOC_VerificationStatusCallout]

5.4.2.2 SecOC_VerificationStatusCalloutWithSecuredPdu

Prototype	
`void` **`SecOC_VerificationStatusCalloutWithSecuredPdu`** `(SecOC_VerificationStatusType verificationStatus, const PduInfoType* receivedSecuredPDU, uint8* fullfreshnessValue, uint32 fullfreshnessValueLength)`	
Parameter	
SecOC_VerificationStatusType	verificationStatus
Const PduInfoType* receivedSecuredPDU	receivedSecuredPDU
uint8*	fullfreshnessValue

uint32	fullfrsehnessValueLength
Return code	
void	none
Functional Description	
Service is used to propagate the status of each verification attempt, the current PDU info and the current freshness value with freshness value length from the SecOC module to other modules. This service can be configured such that: > Only: “False” Verification Status is propagated to modules > Both: “True” and “False” Verification Status is propagated to modules > None: “No Verification Status is propagated“	
Particularities and Limitations	
> none	
Call context	
> task context	

Table 5-20 [SecOC_VerificationStatusCalloutWithSecuredPDU]

5.4.2.3 SecOC_GetTxFreshnessTruncData

Prototype	
`Std_ReturnType` **`SecOC_GetTxFreshnessTruncData`** `(uint16 SecOCFreshnessValueID, uint8* SecOCFreshnessValue, uint32* SecOCFreshnessValueLength, uint8* SecOCTruncatedFreshnessValue, uint32* SecOCTruncatedFreshnessValueLength)`	
Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
SecOCFreshnessValueLength	Holds the length of the provided freshness in bits.
SecOCTruncatedFreshnessValueLength	Provides the truncated freshness length configured for this freshness. The function may adapt the value if needed or can leave it unchanged if the configured length and provided length is the same.
SecOCFreshnessValue	Holds the current freshness value
SecOCTruncatedFreshnessValue	Holds the truncated freshness to be included into the Secured I-PDU. The parameter is optional.
Return code	
Std_ReturnType	E_OK: request successful E_NOT_OK: request failed, a freshness value cannot be provided due to general issues for freshness or this FreshnessValueId. E_BUSY: The freshness information can temporarily not be provided
Functional Description	
This interface is used by the SecOC to obtain the current freshness value. The interface function provides also the truncated freshness transmitted in the secured I-PDU.	

Particularities and Limitations
> none
Call context
> task context

Table 5-21 [SecOC_GetTxFreshnessTruncData]

5.4.2.4 SecOC_GetTxFreshness

Prototype	
`Std_ReturnType SecOC_GetTxFreshness (uint16 SecOCFreshnessValueID, uint8* SecOCFreshnessValue, uint32* SecOCFreshnessValueLength)`	
Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
SecOCFreshnessValueLength	Holds the length of the provided freshness in bits.
SecOCFreshnessValue	Holds the current freshness value
Return code	
Std_ReturnType	E_OK: request successful E_NOT_OK: request failed, a freshness value cannot be provided due to general issues for freshness or this FreshnessValueId. E_BUSY: The freshness information can temporarily not be provided
Functional Description	
This interface is used by the SecOC to obtain the current freshness value.	
Particularities and Limitations	
> none	
Call context	
> task context	

Table 5-22 [SecOC_GetTxFreshness]

5.4.2.5 SecOC_SPduTxConfirmation

Prototype	
`void SecOC_SPduTxConfirmation (uint16 SecOCFreshnessValueID)`	
Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
Return code	
void	none
Functional Description	
This interface is used by the SecOC to indicate that the Secured I-PDU has been initiated for transmission.	

Particularities and Limitations
> none
Call context
> task context

Table 5-23 [SecOC_SPduTxConfirmation]

5.4.2.6 SecOC_GetRxFreshnessAuthData

Prototype

```
Std_ReturnType SecOC_GetRxFreshnessAuthData (uint16 SecOCFreshnessValueID,
    const uint8 *SecOCTruncatedFreshnessValue,
    uint32 SecOCTruncatedFreshnessValueLength,
    const uint8 *SecOCAuthDataFreshnessValue,
    uint16 SecOCAuthDataFreshnessValueLength
    uint16 SecOCAuthVerifyAttempts,
    uint8 *SecOCFreshnessValue,
    uint32 *SecOCFreshnessValueLength)
```

Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
SecOCTruncatedFreshnessValue	Holds the truncated freshness value that was contained in the Secured I-PDU.
SecOCTruncatedFreshnessValueLength	Holds the length in bits of the truncated freshness value.
SecOCAuthDataFreshnessValue	The parameter holds a part of the received, not yet authenticated PDU.
SecOCAuthDataFreshnessValueLength	This is the length value in bits that holds the freshness from the authentic PDU. The parameter is optional (see description).
SecOCAuthVerifyAttempts	Holds the number of authentication verify attempts of this PDU since the last reception. The value is 0 for the first attempt and incremented on every unsuccessful verification attempt.
SecOCFreshnessValue	Holds the length in bits of the freshness value.
SecOCFreshnessValueLength	Holds the freshness value to be used for the calculation of the authenticator.
Return code	
Std_ReturnType	E_OK: request successful E_NOT_OK: request failed, a freshness value cannot be provided due to general issues for freshness or this FreshnessValueId. E_BUSY: The freshness information can temporarily not be provided.
Functional Description	
This interface is used by the SecOC to obtain the current freshness value.	
Particularities and Limitations	
> None	

Call context
> task context

Table 5-24 [SecOC_GetRxFreshnessAuthData]

5.4.2.7 SecOC_GetRxFreshness

Prototype	
`Std_ReturnType SecOC_GetRxFreshness (uint16 SecOCFreshnessValueID, const uint8 *SecOCTruncatedFreshnessValue, uint32 SecOCTruncatedFreshnessValueLength, uint16 SecOCAuthVerifyAttempts, uint8 *SecOCFreshnessValue, uint32 *SecOCFreshnessValueLength)`	
Parameter	
SecOCFreshnessValueID	Holds the identifier of the freshness value.
SecOCTruncatedFreshnessValue	Holds the truncated freshness value that was contained in the Secured I-PDU.
SecOCTruncatedFreshnessValueLength	Holds the length in bits of the truncated freshness value.
SecOCAuthVerifyAttempts	Holds the number of authentication verify attempts of this PDU since the last reception. The value is 0 for the first attempt and incremented on every unsuccessful verification attempt.
SecOCFreshnessValue	Holds the length in bits of the freshness value.
SecOCFreshnessValueLength	Holds the freshness value to be used for the calculation of the authenticator.
Return code	
Std_ReturnType	E_OK: request successful E_NOT_OK: request failed, a freshness value cannot be provided due to general issues for freshness or this FreshnessValueId. E_BUSY: The freshness information can temporarily not be provided.
Functional Description	
This interface is used by the SecOC to obtain the current freshness value.	
Particularities and Limitations	
> None	
Call context	
> task context	

Table 5-25 [SecOC_GetRxFreshness]

5.4.2.8 SecOC_GenerateAuthenticationInfo

Prototype	
`Std_ReturnType SecOC_GenerateAuthenticationInfo(uint16 txFreshnessId, const PduInfoType* txAuthenticPduInfoPtr, uint8* txAuthenticatorPtr, uint32 txAuthenticatorLength )`	
Parameter	
txFreshnessId	Id of the freshness value.
txAuthenticPduInfoPtr	Pointer to the authentic Pdu which shall be used as input for the Authenticator calculation.
txAuthenticatorPtr	Pointer to the authenticator buffer. Write the calculated authenticator to this buffer.
txAuthenticatorLength	Length of the authenticator buffer.
Return code	
Std_ReturnType	E_OK Calculation successful. E_NOT_OK Calculation failed. SECOC_E_BUSY Calculation can not be executed at this moment. The SecOC will call the API again depending on the Build Attempts counter.
Functional Description	
Custom callout for calculating the authenticator value	
Particularities and Limitations	
> None	
Call context	
> task context	

Table 5-26 [SecOC_GenerateAuthenticationInfo]

5.4.2.9 SecOC_VerifyAuthenticationInfo

Prototype	
`Std_ReturnType SecOC_VerifyAuthenticationInfo(uint16 txFreshnessId, const PduInfoType* rxAuthenticPduInfoPtr, const uint8* rxAuthenticatorPtr, uint32 rxAuthenticatorLength, Crypto_VerifyResultType *verifyResultPtr )`	
Parameter	
txFreshnessId	Id of the freshness value.
rxAuthenticPduInfoPtr	Pointer to the authentic Pdu.
rxAuthenticatorPtr	Pointer to the authenticator buffer.
rxAuthenticatorLength	Length of the authenticator buffer.

verifyResultPtr	Pointer to result buffer.
Return code	
Std_ReturnType	E_OK Verification successful. E_NOT_OK Verification failed. SECOC_E_BUSY Verification cannot be executed at this moment. The SecOC will call the API again depending on the Build Attempts counter.
Functional Description	
Custom callout for the verification of the received secured Pdu	
Particularities and Limitations	
> None	
Call context	
> task context	

Table 5-27 [SecOC_VerifyAuthenticationInfo]

5.5 Service Ports

5.5.1 Sender Receiver Interface

Operation	Notification
{SecOC_VerificationStatusService}	Is used to propagate the status of each authentication attempt.

5.5.2 Client Server Interface

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

5.5.2.1 Provide Ports on SecOC Side

At the Provide Ports of the SecOC the API functions described in 5.1 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.

The following sub-chapters present the Provide Ports defined for the SecOC and the Operations defined for the Provide Ports, the API functions related to the Operations to be added by the RTE.

5.5.2.1.1 [Provide Port]

Operation	API Function	Port Defined Argument Values
SecOCVerifyStatusOverride	SecOC_VerifyStatusOverride	uint16 freshnessValueId, uint8 overrideStatus, uint8 numberOfMessagesToOverride

Table 5-28 [Provide Port]

5.5.2.2 Require Ports on SecOC Side

At its Require Ports the SecOC calls Operations. These Operations have to be provided by the SWCs by means of Runnable Entities. These Runnable Entities implement the callback functions expected by the SecOC.

The following sub-chapters present the Require Ports defined for the SecOC, the Operations that are called from the SecOC and the related Notifications, which are described in chapter 5.4.

5.5.2.2.1 [Require Port]

Operation	Notification
{SecOC_GetTxFreshnessTruncData}	This operation is used by the SecOC to obtain the freshness that corresponds to the freshnessValueId. The operation provides the freshness and also the truncated freshness that shall be placed into the Secured-IPDU.
{GetTxFreshness}	This operation is used by the SecOC to obtain the freshness that corresponds to the freshnessValueId.
{SPduTxConfirmation}	This operation is used by the SecOC to indicate that the Secured I-PDU has been initiated for transmission.

Operation	Notification
{GetRxFreshness}	This interface is used by the SecOC to obtain the current freshness value. This operation provides also a part of the Authentic-PDU data if configured.
{GetRxFreshnessAuthData}	This interface is used by the SecOC to obtain the current freshness value. This operation provides also a part of the Authentic-PDU data if configured.
VerifyStatus	This service provides the ability to inform the application about the result of the verification attempt of a received PDU by the SecOC module.

Table 5-29 [Require Port]

6 Configuration

6.1 Configuration Variants

The SecOC supports the configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-POST-BUILD-LOADABLE
- VARIANT- POST-BUILD-SELECTABLE

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

7 Glossary

7.1 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
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
PDU	Protocol data unit
I-PDU	Interface protocol data unit

Table 7-1 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com