Document Title	Specification of Module Secure Onboard Communication
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	654
Document Classification	Standard

Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	Changed by	Change Description
4.2.2	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
4.2.1	AUTOSAR Release Management	• Initial Release

Disclaimer

Advice for users

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

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

Table of contents

1 Introduction and functional overview

This specification is the AUTOSAR Secure Onboard Communication (SecOC) module Software Specification. It is based on AUTOSAR SecOC [5] and specifies how the requirements of the AUTOSAR SecOC SRS shall be realized. It describes the basic security features, the functionality and the API of the AUTOSAR SecOC module.

The SecOC module aims for resource-efficient and practicable authentication mechanisms for critical data on the level of PDUs. The authentication mechanisms shall be seamlessly integrated with the current AUTOSAR communication systems. The impact with respect to resource consumption should be as small as possible in order to allow protection as add-on for legacy systems. The specification is based on the assumption that mainly symmetric authentication approaches with message authentication codes (MACs) are used. They achieve the same level of security with much smaller keys than asymmetric approaches and can be implemented compactly and efficiently in software and in hardware. However, the specification provides the necessary level of abstraction so that both, symmetric approaches as well as asymmetric authentication approaches can be used.

The SecOC module integrates on the level of the AUTOSAR PduR. **Figure 1** shows the integration of the SecOC module as part of the Autosar communication stack.

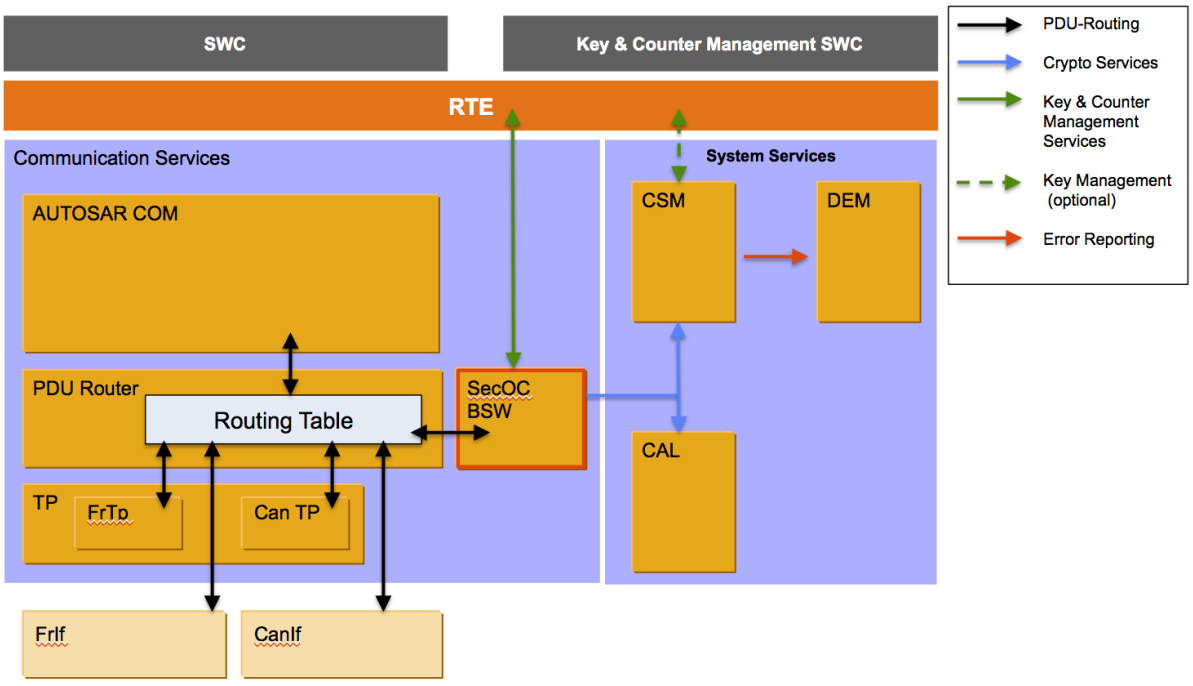

Figure 1: Integration of the SecOC BSW

In this setting, PduR is responsible to route incoming and outgoing security related I-PDUs to the SecOC module. The SecOC module shall then add or process the security relevant information and shall propagate the results in the form of an I-PDU back to the PduR. PduR is then responsible to further route the I-PDUs. Moreover, the SecOC module makes use of the cryptographic services provided by either the CSM or the CAL and interacts with the RTE to allow key and counter management.

The SecOC module shall support all kind of communication paradigms and principles that are supported by PduR, especially Multicast communications, Transport Protocols and the PduR Gateway. However, since the SecOC module is restricted to the IF API towards the upper layer, the authentication of PDUs from and to DCM/J1939DCM is currently not supported. The following sections provide a detailed specification of SecOC interfaces, functionality and configuration.

2 Acronyms, abbreviations and definitions

2.1 Acronyms and abbreviations

Abbreviation / Acronym:	*Description:*
CAL	The AUTOSAR Crypto Abstraction Layer
CSM	The AUTOSAR Crypto Service Manager
SecOC	Secure Onboard Communication
MAC	Message Authentication Code

2.2 Definitions

For this document the definitions of data integrity, authentication, entity authentication, data origin, message authentication and transaction authentication from [14] are used:

Term:	*Description:*
Authentic I-PDU	An Authentic I-PDU is an arbitrary AUTOSAR I-PDU that is completely secured during network transmission by means of the Secured I-PDU
Authentication	Authentication is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons, this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).
Authentication Information	The Authentication Information consists of a Freshness Value (or a part thereof) and an Authenticator (or a part thereof). Authentication Information are the additional pieces of information that are added by SecOC to realize the Secured I-PDU
Authenticator	Authenticator is data that is used to provide message authentication. In general, the term Message Authentication Code (MAC) is used for symmetric approaches while the term Signature or Digital Signature refers to asymmetric approaches having different properties and constraints.
Data integrity	Data integrity is the property whereby data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source. To assure data integrity, one should have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.
Data origin authentication	Data origin authentication is a type of authentication whereby a party is corroborated as the (original) source of specified data

	created at some (typically unspecified) time in the past. By definition, data origin authentication includes data integrity.
Distinction unilateral/ bilateral authentication	In unilateral authentication, one side proves identity. The requesting side is not even authenticated to the extent of proving that it is allowed to request authentication. In bilateral authentication, the requester is also authenticated at least (see below) to prove the privilege of requesting. There is an efficient and more secure way to authenticate both endpoints, based on the bilateral authentication described above. Along with the authentication (in the second message) requested initially by the receiver (in the first message), the sender also requests an authentication. The receiver sends a third message providing the authentication requested by the sender. This is only three messages (in contrast to four with two unilateral messages).
Entity authentication	Entity authentication is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired). Note: Entity authentication means to prove presence and operational readiness of a communication endpoint. This is for example often done by proving access to a cryptographic key and knowledge of a secret. It is necessary to do this without disclosing either key or secret. Entity authentication can be used to prevent record-and-replay attacks. Freshness of messages only complicates them by the need to record a lifetime and corrupt either senders or receivers (real-time) clock. Entity authentication is triggered by the receiver, i.e. the one to be convinced, while the sender has to react by convincing. Record and replay attacks on entity authentication are usually prevented by allowing the receiver some control over the authentication process. In order to prevent the receiver from using this control for steering the sender to malicious purposes or from determining a key or a secret ("oracle attack"), the sender can add more randomness. If not only access to a key (implying membership to a privileged group) but also individuality is to be proven, the sender additionally adds and authenticates its unique identification.
Message authentication	Message authentication is a term used analogously with data origin authentication. It provides data origin authentication with respect to the original message source (and data integrity, but no uniqueness and timeliness guarantees).
Secured I-PDU	A Secured I-PDU is an AUTOSAR I-PDU that contains Payload of an Authentic I-PDU supplemented by additional Authentication Information.
Transaction authentication	Transaction authentication denotes message authentication augmented to additionally provide uniqueness and timeliness guarantees on data (thus preventing undetectable message replay).

3 Related documentation

3.1 Input documents

[1] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] AUTOSAR General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] AUTOSAR General Specification for Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

[4] Specification of Communication
AUTOSAR_SWS_COM - Specification of Communication

[5] AUTOSAR SecOC Software Requirements Specification
AUTOSAR_SRS_SecureOnboardCommunication.pdf

[6] Specification of I-PDU Multiplexer
AUTOSAR_SWS_I-PDUMultiplexer.pdf

[7] Specification of PDU Router
AUTOSAR_SWS_PduRouter.pdf

[8] Specification of Crypt Service Manager
AUTOSAR_SWS_CryptoServiceManager.pdf

[9] System Template,
https://svn3.autosar.org/repos2/work/24_Sources/branches/R4.0/TPS_SystemTemplate_063/AUTOSAR_TPS_SystemTemplate.pdf

[10] Software Component Template,
https://svn3.autosar.org/repos2/work/24_Sources/branches/R4.0/TPS_SoftwareComponentTemplate_062/AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[11] Koscher et al: Experimental Security Analysis of a Modern Automobile, 2010 IEEE Symposium on Security and Privacy

[12] Checkoway et al: Comprehensive Experimental Analyses of Automotive Attack Surfaces, USENIX Security 2011

[13] Auguste Kerckhoffs, ‘La cryptographie militaire’, Journal des sciences militaires, vol. IX, pp. 5–38, Jan. 1883, pp. 161–191, Feb. 1883.

[14] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[15] Danny Dolev and Andrew C. Yao: On the security of public key protocols, In Foundations of Computer Science, SFCS 1981

[16] M. Dworkin: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, U.S. Department of Commerce, Information Technology Laboratory (ITL), National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, NIST Special Publication 800-38B, 2005

3.2 Related standards and norms

[17] IEC 7498-1 The Basic Model, IEC Norm, 1994

[18] National Institute of Standards and Technology (NIST): FIPS-180-4, Secure Hash Standard (SHS), March 2012, available electronically at http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[19] FIPS Pub 197: Advanced Encryption Standard (AES), U.S. Department of Commerce, Information Technology Laboratory (ITL), National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, Federal Information Processing Standards Publication, 2001, electronically available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software (SWS BSW General) [3], which is also valid for SecOC module

Thus, the SWS BSW General specification [3] shall be considered as an additional set of requirements for the AUTOSAR SecOC module.

4 Constraints and assumptions

This document is applicable for AUTOSAR release 4.2.

4.1 Limitations

The SecOC module is restricted to only provide the IF API towards the upper layer. Thus, the authentication of PDUs from and to DCM/J1939DCM is currently not supported.

4.2 Applicability to car domains

The SecOC module is used in all ECUs where secure communication is necessary.

The SecOC module has not been specified to work with MOST and LIN communication networks. With MOST not being specifically supported, the applicability to multimedia and telematic car domains may be limited.

5 Dependencies to other modules

This chapter lists all the features from other modules that are used by the AUTOSAR SecOC module and functionalities that are provided by the AUTOSAR SecOC module to other modules. Because the SecOC module deals with I-PDUs that are either sourced or sunk by other modules, care should be taken that shared configuration items are consistent between the modules.

5.1 Dependencies to PduR

The SecOC module depends on the API and capabilities of the PduR. It provides the upper and lower layer API functions required by the PDU Router, namely

- the API of the communication interface modules,
- the API of the Transport Protocol Modules,
- the API of the upper layer modules which use transport protocol modules,
- the API of the upper layer modules which process I-PDUs originating from communication interface modules.

To serve the PduR with the results of the security processing, the SecOC module requires the respective API function of the PduR.

5.2 Dependencies to CSM or CAL

The SecOC module depends on cryptographic algorithms that are provided in AUTOSAR by either the CSM module or the CAL module. The SecOC module requires API functions to generate and verify Cryptographic Signatures or Message Authentication Codes, namely

- the MAC-generate interface (<Csm/Cal>_MacGenerate<Start/Update/Finish>),
- the MAC-verify interface (<Csm/Cal>_MacVerify<Start/Update/Finish>),
- the Signature-generate interface (<Csm/Cal>_SignatureGenerate<Start/Update/Finish>),
- the Signature-verify interface (<Csm/Cal>_SignatureVerify<Start/Update/Finish>),

5.3 Dependencies to the RTE

The SecOC module provides an API to manage keys and freshness values. This API contains the following API functions that are provided as Service Interfaces by the RTE.

- SecOC_AssociateKey,
- SecOC_FreshnessValueRead,
- SecOC_FreshnessValueWrite,
- SecOC_VerificationStatus,
- SecOC_VerifyStatusOverride.

The API functions are specified in more detail in Section 8.

The RTE includes the BSW-Scheduler. The SecOC module relies on the BSW-scheduler calling the SecOC_MainFunction function at a period as configured in SecOCMainFunctionPeriod.

5.4 File structure

5.4.1 Header file structure

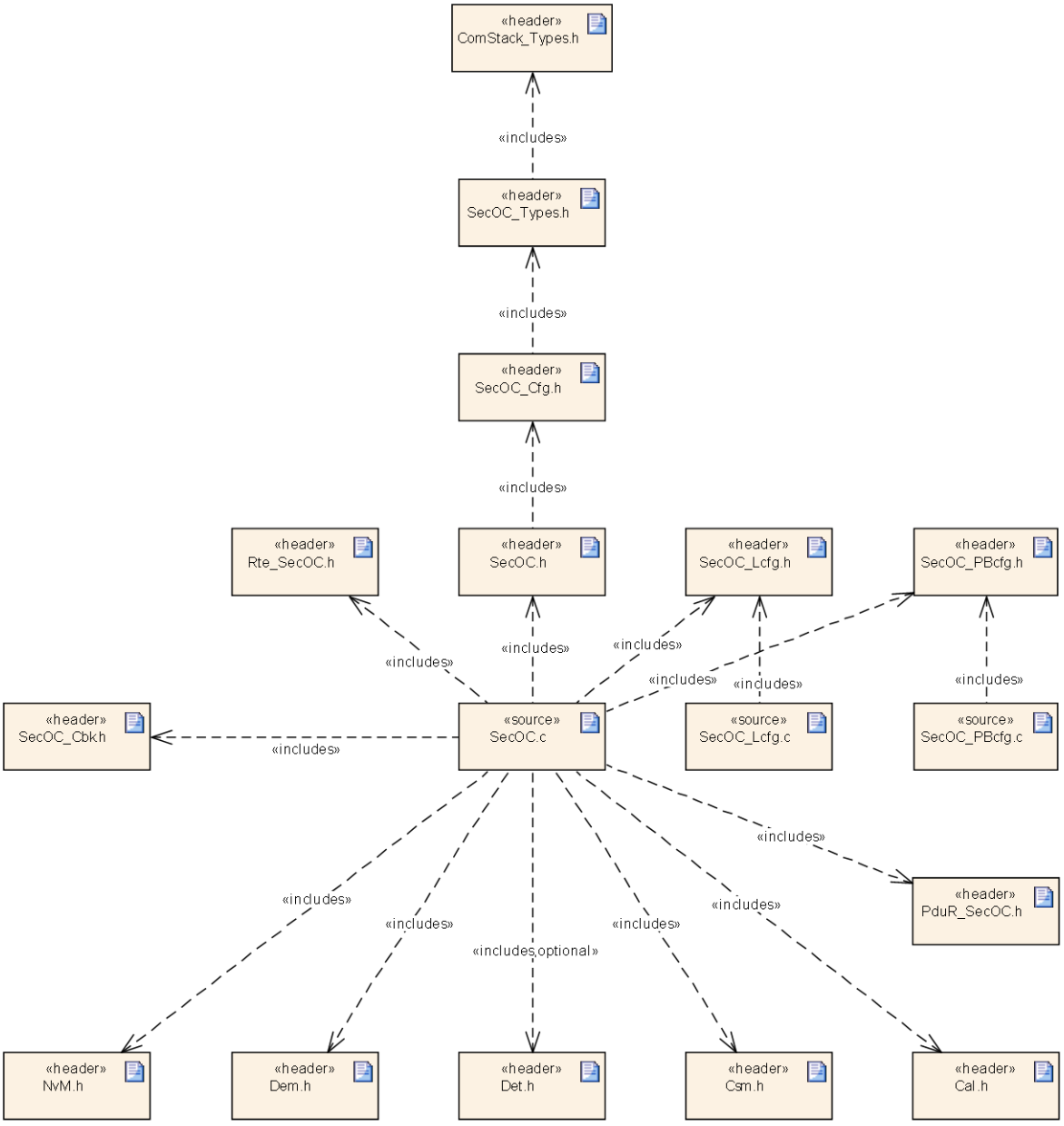

Figure 2: SecOC header file structure

[SWS_SecOC_00001]⌈
General SecOC module definitions shall be defined in SecOC.h.
⌋ (SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00381, SRS_BSW_00415)

[SWS_SecOC_00002]⌈
Type definitions of the SecOC module shall be defined in SecOC_Types.h.
⌋ (SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00381, SRS_BSW_00415)

6 Requirements traceability

The following table references the requirements specified in [3] and [5] and links to the fulfillment of these.

Requirement	Description	Satisfied by
-	-	SWS_SecOC_00106
-	-	SWS_SecOC_00147
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_SecOC_00107
SRS_BSW_00004	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	SWS_SecOC_00999
SRS_BSW_00005	Modules of the ÂµC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_SecOC_00999
SRS_BSW_00006	The source code of software modules above the ÂµC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_SecOC_00999
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	SWS_SecOC_00999
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_SecOC_00999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_SecOC_00999
SRS_BSW_00158	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation	SWS_SecOC_00999
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_SecOC_00143, SWS_SecOC_00144
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_SecOC_00999
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_SecOC_00999
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_SecOC_00999
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_SecOC_00999
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_SecOC_00999
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_SecOC_00999
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_SecOC_00999
SRS_BSW_00171	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	SWS_SecOC_00153

SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_SecOC_00999
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_SecOC_00999
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_SecOC_00103
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_SecOC_00999
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	SWS_SecOC_00999
SRS_BSW_00305	Data types naming convention	SWS_SecOC_00999
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_SecOC_00999
SRS_BSW_00307	Global variables naming convention	SWS_SecOC_00999
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_SecOC_00999
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_SecOC_00999
SRS_BSW_00310	API naming convention	SWS_SecOC_00999
SRS_BSW_00312	Shared code shall be reentrant	SWS_SecOC_00999
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_SecOC_00999
SRS_BSW_00318	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	SWS_SecOC_00999
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_SecOC_00999
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_SecOC_00107, SWS_SecOC_00112, SWS_SecOC_00113, SWS_SecOC_00116, SWS_SecOC_00117, SWS_SecOC_00118, SWS_SecOC_00122, SWS_SecOC_00124, SWS_SecOC_00125, SWS_SecOC_00126, SWS_SecOC_00127, SWS_SecOC_00128, SWS_SecOC_00129, SWS_SecOC_00130, SWS_SecOC_00152, SWS_SecOC_00156, SWS_SecOC_00157, SWS_SecOC_00161
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_SecOC_00999
SRS_BSW_00327	Error values naming convention	SWS_SecOC_00999
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the	SWS_SecOC_00999

	duplication of code	
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_SecOC_00999
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_SecOC_00999
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_SecOC_00999
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_SecOC_00999
SRS_BSW_00335	Status values naming convention	SWS_SecOC_00999
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_SecOC_00999
SRS_BSW_00337	Classification of development errors	SWS_SecOC_00101, SWS_SecOC_00102, SWS_SecOC_00164, SWS_SecOC_00165, SWS_SecOC_00166, SWS_SecOC_00167
SRS_BSW_00339	Reporting of production relevant error status	SWS_SecOC_00999
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_SecOC_00999
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_SecOC_00999
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_SecOC_00999
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_SecOC_00144
SRS_BSW_00345	BSW Modules shall support pre-compile configuration	SWS_SecOC_00143
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	SWS_SecOC_00999
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_SecOC_00999
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_SecOC_00001, SWS_SecOC_00002
SRS_BSW_00350	All AUTOSAR Basic Software Modules shall apply a specific naming rule for enabling/disabling the detection and reporting of development errors	SWS_SecOC_00102, SWS_SecOC_00164, SWS_SecOC_00165, SWS_SecOC_00166, SWS_SecOC_00167
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_SecOC_00001, SWS_SecOC_00002
SRS_BSW_00357	For success/failure of an API call a standard return type shall be defined	SWS_SecOC_00112, SWS_SecOC_00113, SWS_SecOC_00116, SWS_SecOC_00117, SWS_SecOC_00118, SWS_SecOC_00122, SWS_SecOC_00127, SWS_SecOC_00128,

		SWS_SecOC_00129, SWS_SecOC_00130
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_SecOC_00107, SWS_SecOC_00119, SWS_SecOC_00124, SWS_SecOC_00125, SWS_SecOC_00126, SWS_SecOC_00152, SWS_SecOC_00161
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_SecOC_00999
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_SecOC_00999
SRS_BSW_00369	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	SWS_SecOC_00107, SWS_SecOC_00112
SRS_BSW_00371	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	SWS_SecOC_00999
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_SecOC_00131
SRS_BSW_00374	All Basic Software Modules shall provide a readable module vendor identification	SWS_SecOC_00999
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_SecOC_00999
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_SecOC_00999
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_SecOC_00999
SRS_BSW_00379	All software modules shall provide a module identifier in the header file and in the module XML description file.	SWS_SecOC_00999
SRS_BSW_00380	Configuration parameters being stored in memory shall be placed into separate c-files	SWS_SecOC_00999
SRS_BSW_00381	The pre-compile time parameters shall be placed into a separate configuration header file	SWS_SecOC_00001, SWS_SecOC_00002
SRS_BSW_00383	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	SWS_SecOC_00999
SRS_BSW_00384	The Basic Software Module specifications shall specify at least in the description which other modules they require	SWS_SecOC_00137, SWS_SecOC_00138
SRS_BSW_00385	List possible error notifications	SWS_SecOC_00077, SWS_SecOC_00089, SWS_SecOC_00101, SWS_SecOC_00102, SWS_SecOC_00108, SWS_SecOC_00109, SWS_SecOC_00121, SWS_SecOC_00151, SWS_SecOC_00155, SWS_SecOC_00164, SWS_SecOC_00165, SWS_SecOC_00166,

		SWS_SecOC_00167
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_SecOC_00101
SRS_BSW_00388	Containers shall be used to group configuration parameters that are defined for the same object	SWS_SecOC_00999
SRS_BSW_00389	Containers shall have names	SWS_SecOC_00999
SRS_BSW_00390	Parameter content shall be unique within the module	SWS_SecOC_00999
SRS_BSW_00392	Parameters shall have a type	SWS_SecOC_00999
SRS_BSW_00393	Parameters shall have a range	SWS_SecOC_00999
SRS_BSW_00394	The Basic Software Module specifications shall specify the scope of the configuration parameters	SWS_SecOC_00999
SRS_BSW_00395	The Basic Software Module specifications shall list all configuration parameter dependencies	SWS_SecOC_00999
SRS_BSW_00396	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container	SWS_SecOC_00999
SRS_BSW_00397	The configuration parameters in pre-compile time are fixed before compilation starts	SWS_SecOC_00999
SRS_BSW_00398	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	SWS_SecOC_00999
SRS_BSW_00399	Parameter-sets shall be located in a separate segment and shall be loaded after the code	SWS_SecOC_00999
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	SWS_SecOC_00999
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_SecOC_00999
SRS_BSW_00402	Each module shall provide version information	SWS_SecOC_00107
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_SecOC_00145
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_SecOC_00999
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_SecOC_00999
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_SecOC_00107
SRS_BSW_00408	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	SWS_SecOC_00999
SRS_BSW_00409	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	SWS_SecOC_00999
SRS_BSW_00410	Compiler switches shall have defined values	SWS_SecOC_00999
SRS_BSW_00411	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	SWS_SecOC_00999
SRS_BSW_00412	References to c-configuration parameters shall be placed into a separate h-file	SWS_SecOC_00999

SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_SecOC_00999
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_SecOC_00001, SWS_SecOC_00002
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_SecOC_00999
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_SecOC_00999
SRS_BSW_00419	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	SWS_SecOC_00999
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_SecOC_00999
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_SecOC_00999
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_SecOC_00999
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_SecOC_00131
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_SecOC_00110, SWS_SecOC_00111
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_SecOC_00999
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_SecOC_00999
SRS_BSW_00429	BSW modules shall be only allowed to use OS objects and/or related OS services	SWS_SecOC_00999
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_SecOC_00999
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_SecOC_00999
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_SecOC_00999
SRS_BSW_00438	Configuration data shall be defined in a structure	SWS_SecOC_00999
SRS_BSW_00439	Enable BSW modules to handle interrupts	SWS_SecOC_00999
SRS_BSW_00440	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	SWS_SecOC_00999
SRS_BSW_00441	Naming convention for type, macro and function	SWS_SecOC_00999
SRS_BSW_00442	{OBSOLETE} The AUTOSAR architecture shall support standardized debugging and tracing features	SWS_SecOC_00999
SRS_BSW_00447	Standardizing Include file structure of BSW Modules Implementing Autosar Service	SWS_SecOC_00999
SRS_BSW_00448	Module SWS shall not contain requirements from Other Modules	SWS_SecOC_00999

SRS_BSW_00449	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	SWS_SecOC_00112, SWS_SecOC_00113, SWS_SecOC_00116, SWS_SecOC_00117, SWS_SecOC_00118, SWS_SecOC_00122, SWS_SecOC_00125, SWS_SecOC_00127, SWS_SecOC_00152
SRS_BSW_00450	A Main function of a un-initialized module shall return immediately	SWS_SecOC_00102
SRS_BSW_00451	Hardware registers shall be protected if concurrent access to these registers occur	SWS_SecOC_00999
SRS_BSW_00452	Classification of runtime errors	SWS_SecOC_00999
SRS_BSW_00453	BSW Modules shall be harmonized	SWS_SecOC_00999
SRS_BSW_00454	An alternative interface without a parameter of category DATA_REFERENCE shall be available.	SWS_SecOC_00999
SRS_BSW_00456	- A Header file shall be defined in order to harmonize BSW Modules	SWS_SecOC_00999
SRS_BSW_00457	- Callback functions of Application software components shall be invoked by the Basis SW	SWS_SecOC_00012
SRS_BSW_00458	Classification of production errors	SWS_SecOC_00999
SRS_BSW_00459	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	SWS_SecOC_00999
SRS_BSW_00460	Reentrancy Levels	SWS_SecOC_00999
SRS_BSW_00461	Modules called by generic modules shall satisfy all interfaces requested by the generic module	SWS_SecOC_00999
SRS_BSW_00462	All Standardized Autosar Interfaces shall have unique requirement Id / number	SWS_SecOC_00999
SRS_BSW_00463	Naming convention of callout prototypes	SWS_SecOC_00999
SRS_BSW_00464	File names shall be considered case sensitive regardless of the filesystem in which they are used	SWS_SecOC_00999
SRS_BSW_00465	It shall not be allowed to name any two files so that they only differ by the cases of their letters	SWS_SecOC_00999
SRS_BSW_00466	Classification of extended production errors	SWS_SecOC_00999
SRS_BSW_00467	The init / deinit services shall only be called by BswM or EcuM	SWS_SecOC_00999
SRS_BSW_00469	Fault detection and healing of production errors and extended production errors	SWS_SecOC_00999
SRS_BSW_00470	Execution frequency of production error detection	SWS_SecOC_00999
SRS_BSW_00471	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	SWS_SecOC_00999
SRS_BSW_00472	Avoid detection of two production errors with the same root cause.	SWS_SecOC_00999
SRS_SecOC_00001	Selection of Authentic I-PDU [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00104
SRS_SecOC_00002	Range of verification retry by the receiver	SWS_SecOC_00015,

	[open/proposed/conflicts/approved/rejected]	SWS_SecOC_00022, SWS_SecOC_00023, SWS_SecOC_00024, SWS_SecOC_00045, SWS_SecOC_00047, SWS_SecOC_00049, SWS_SecOC_00052, SWS_SecOC_00053, SWS_SecOC_00091, SWS_SecOC_00092, SWS_SecOC_00093, SWS_SecOC_00094, SWS_SecOC_00117, SWS_SecOC_00118, SWS_SecOC_00140, SWS_SecOC_00168
SRS_SecOC_00003	Configuration of different security properties / requirements [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00012, SWS_SecOC_00104, SWS_SecOC_00116, SWS_SecOC_00139
SRS_SecOC_00005	Initialisation of security information [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00054, SWS_SecOC_00105, SWS_SecOC_00132, SWS_SecOC_00154, SWS_SecOC_00162
SRS_SecOC_00006	Creation of a Secured I-PDU from an Authentic I-PDU [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00011, SWS_SecOC_00031, SWS_SecOC_00033, SWS_SecOC_00034, SWS_SecOC_00035, SWS_SecOC_00036, SWS_SecOC_00037, SWS_SecOC_00038, SWS_SecOC_00040, SWS_SecOC_00042, SWS_SecOC_00046, SWS_SecOC_00057, SWS_SecOC_00058, SWS_SecOC_00146, SWS_SecOC_00156, SWS_SecOC_00157, SWS_SecOC_00161
SRS_SecOC_00007	Verification retry by the receiver [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00015, SWS_SecOC_00017, SWS_SecOC_00018, SWS_SecOC_00019, SWS_SecOC_00020, SWS_SecOC_00021, SWS_SecOC_00022, SWS_SecOC_00023, SWS_SecOC_00024, SWS_SecOC_00028, SWS_SecOC_00045, SWS_SecOC_00047, SWS_SecOC_00049, SWS_SecOC_00052, SWS_SecOC_00053, SWS_SecOC_00091,

		SWS_SecOC_00092, SWS_SecOC_00093, SWS_SecOC_00094, SWS_SecOC_00168
SRS_SecOC_00010	Communication security is available for all communication paradigms of AUTOSAR [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00060, SWS_SecOC_00061, SWS_SecOC_00062, SWS_SecOC_00063, SWS_SecOC_00064, SWS_SecOC_00065, SWS_SecOC_00066, SWS_SecOC_00067, SWS_SecOC_00068, SWS_SecOC_00069, SWS_SecOC_00070, SWS_SecOC_00071, SWS_SecOC_00072, SWS_SecOC_00073, SWS_SecOC_00074, SWS_SecOC_00075, SWS_SecOC_00078, SWS_SecOC_00079, SWS_SecOC_00080, SWS_SecOC_00082, SWS_SecOC_00083, SWS_SecOC_00084, SWS_SecOC_00085, SWS_SecOC_00086, SWS_SecOC_00150
SRS_SecOC_00012	Support of Automotive BUS Systems [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00060, SWS_SecOC_00061, SWS_SecOC_00062, SWS_SecOC_00063, SWS_SecOC_00064, SWS_SecOC_00065, SWS_SecOC_00066, SWS_SecOC_00067, SWS_SecOC_00068, SWS_SecOC_00069, SWS_SecOC_00070, SWS_SecOC_00071, SWS_SecOC_00072, SWS_SecOC_00073, SWS_SecOC_00074, SWS_SecOC_00075, SWS_SecOC_00078, SWS_SecOC_00079, SWS_SecOC_00080, SWS_SecOC_00082, SWS_SecOC_00083, SWS_SecOC_00084, SWS_SecOC_00085, SWS_SecOC_00086, SWS_SecOC_00113, SWS_SecOC_00124, SWS_SecOC_00125, SWS_SecOC_00126, SWS_SecOC_00127, SWS_SecOC_00128,

		SWS_SecOC_00129, SWS_SecOC_00130, SWS_SecOC_00150, SWS_SecOC_00152
SRS_SecOC_00013	Support for end-to-end and point-to-point protection	SWS_SecOC_00060, SWS_SecOC_00061, SWS_SecOC_00062, SWS_SecOC_00063, SWS_SecOC_00064, SWS_SecOC_00065, SWS_SecOC_00066, SWS_SecOC_00067, SWS_SecOC_00068, SWS_SecOC_00069, SWS_SecOC_00070, SWS_SecOC_00071, SWS_SecOC_00072, SWS_SecOC_00073, SWS_SecOC_00074, SWS_SecOC_00075, SWS_SecOC_00078, SWS_SecOC_00079, SWS_SecOC_00080, SWS_SecOC_00082, SWS_SecOC_00083, SWS_SecOC_00084, SWS_SecOC_00085, SWS_SecOC_00086, SWS_SecOC_00150
SRS_SecOC_00017	PDU security information override [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00119, SWS_SecOC_00122, SWS_SecOC_00142
SRS_SecOC_00020	Security operational information persistency[open/proposed/conflicts/approved/rejected]	SWS_SecOC_00019, SWS_SecOC_00055, SWS_SecOC_00156, SWS_SecOC_00161
SRS_SecOC_00021	Transmitted PDU authentication failure handling [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00076, SWS_SecOC_00087, SWS_SecOC_00151
SRS_SecOC_00022	Received PDU verification failure handling[open/proposed/conflicts/approved/rejected]	SWS_SecOC_00045, SWS_SecOC_00047, SWS_SecOC_00048, SWS_SecOC_00050, SWS_SecOC_00052, SWS_SecOC_00053, SWS_SecOC_00087, SWS_SecOC_00121, SWS_SecOC_00141, SWS_SecOC_00148, SWS_SecOC_00149, SWS_SecOC_00160
SRS_SecOC_00025	Authentication and verification processing time [open/proposed/conflicts/approved/rejected]	SWS_SecOC_00133, SWS_SecOC_00134, SWS_SecOC_00135, SWS_SecOC_00136

7 Functional specification

Authentication and integrity protection of sensitive data is necessary to protect correct and safe functionality of the vehicle systems – this ensures that received data comes from the right ECU and has the correct value.

The SecOC module aims for resource-efficient and practicable authentication mechanisms of sensitive data on the level of PDUs. The approach proposed in this specification generally supports the use of symmetric and asymmetric methods for authenticity and integrity protection. Both methods roughly aim at the same goal and show major similarities in the concept, but there are also some differences due to differing technical properties of the underlying primitives. In addition, the commonly used terms for Authenticator are different. In general, the term Message Authentication Code (MAC) is used for symmetric approaches while the term signature or digital signature refers to asymmetric approaches having different properties and constraints.

In order to ease presentation and improve legibility, the following approach is taken: The subsequent section describes the technical approach using symmetric mechanisms in some detail. Here also the common terms for symmetric primitives are used. The adaptations that need to be done in case of an asymmetric approach are separately given in section 7.1.4.

7.1 Specification of the security solution

The SecOC module as described in this document provides functionality necessary to verify the authenticity and freshness of PDU based communication between ECUs within the vehicle architecture. The approach requires both the sending ECU and the receiving ECU to implement a SecOC module. Both SecOC modules are integrated providing the upper and lower layer PduR APIs on the sender and receiver side. The SecOC modules on both sides generally interact with the PduR module.

To provide message freshness, the SecOC module on the sending and receiving side maintains Freshness Values (e.g. Freshness Counter, Timestamp) for each uniquely identifiable Secured I-PDU, i.e. for each secured communication link.

On the sender side, the SecOC module creates a Secured I-PDU by adding authentication information to the outgoing Authentic I-PDU. The authentication information comprises of an Authenticator (e.g. Message Authentication Code) and optionally a Freshness Value. Regardless if the Freshness Value is or is not included in the Secure I-PDU payload, the Freshness Value is considered during generation of the Authenticator. When using a Freshness Counter instead of a Timestamp, the Freshness Counter is incremented prior to providing the authentication information to the receiver side.

On the receiver side, the SecOC module checks the freshness and authenticity of the Authentic I-PDU by verifying the authentication information that has been appended by the sending side SecOC module. To verify the authenticity and freshness of an Authentic I-PDU, the Secured I-PDU provided to the receiving side SecOC should be

the same Secured I-PDU provided by the sending side SecOC and the receiving side SecOC should have knowledge of the Freshness Value used by the sending side SecOC during creation of the Authenticator.

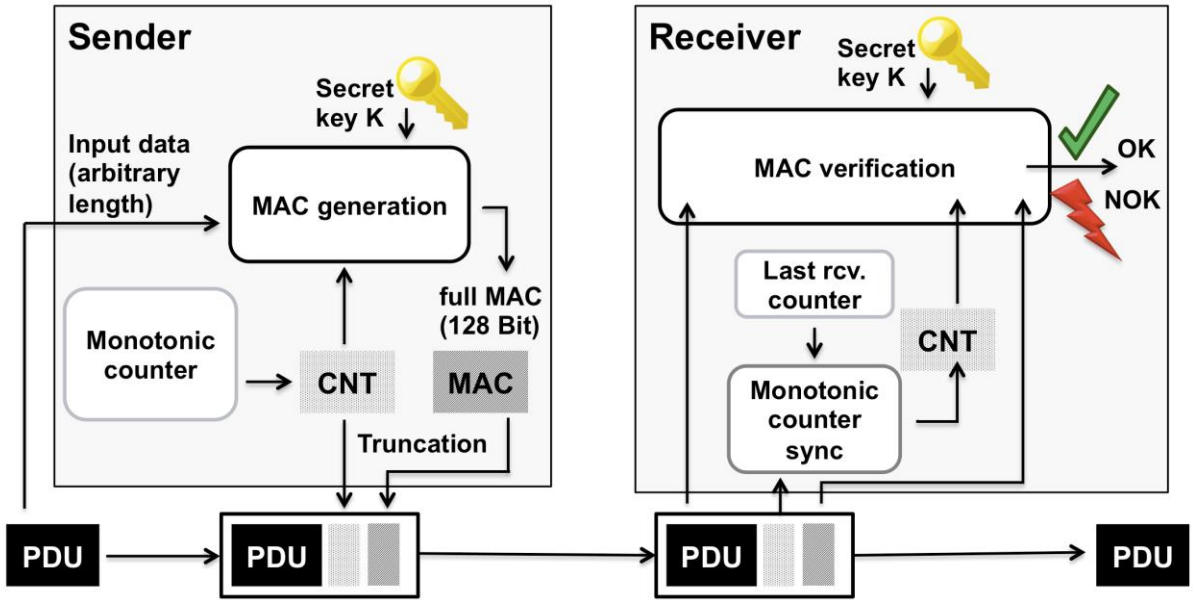

Figure 3: Message Authentication and Freshness Verification

The main purpose of the SecOC module is the realization of the security functionality described throughout this specification.

7.1.1 Basic entities of the security solution

7.1.1.1 Authentic I-PDU and Secured I-PDU

The term Authentic I-PDU refers to an AUTOSAR I-PDU that requires protection against unauthorized manipulation and replay attacks.

The payload of a Secured I-PDU consists of the Authentic I-PDU and an Authenticator (e.g. Message Authentication Code). The payload of a Secured I-PDU may optionally include the Freshness Value used to create the Authenticator (e.g. MAC). The order in which the contents are structured in the Secured I-PDU is compliant with **Figure 4**.

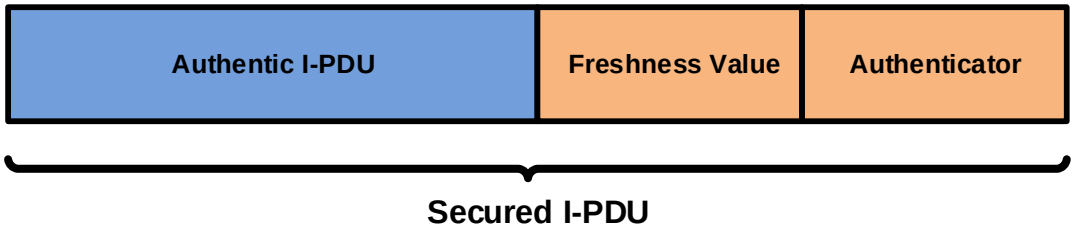

Figure 4: Secured I-PDU contents

The length of the Authentic I-PDU, the Freshness Value and the Authenticator within a Secured I-PDU may vary from one uniquely indefinable Secured I-PDU to another.

The Authenticator (e.g. MAC) refers to a unique authentication data string generated using a Key, Data Identifier of the Secured I-PDU, Authentic Payload, and Freshness Value. The Authenticator provides a high level of confidence that the data in an Authentic I-PDU is generated by a legitimate source and is provided to the receiving ECU at the time in which it is intended for.

Depending on the authentication algorithm (parameter SecOCTxAuthServiceConfigRef or SecOCRxAuthServiceConfigRef) used to generate the Authenticator, it may be possible to truncate the resulting Authenticator (e.g. in case of a MAC) generated by the authentication algorithm. Truncation may be desired when the message payload is limited in length and does not have sufficient space to include the full Authenticator.

The Authenticator length contained in a Secured I-PDU (parameter SecOCAuthInfoTxLength) is specific to a uniquely identifiable Secured I-PDU. This allows provision of flexibility across the system (i.e. two independent unique Secured I-PDUs may have different Authenticator lengths included in the payload of the Secure I-PDU) by providing fine grain configuration of the MAC truncation length for each Secured I-PDU.

If truncation is possible, the Authenticator should only be truncated down to the most significant bits of the resulting Authenticator generated by the authentication algorithm. **Figure 5** shows the truncation of the Authenticator and the Freshness Values respecting the parameter SecOCFreshnessValueTxLength and SecOCAuthInfoTxLength.

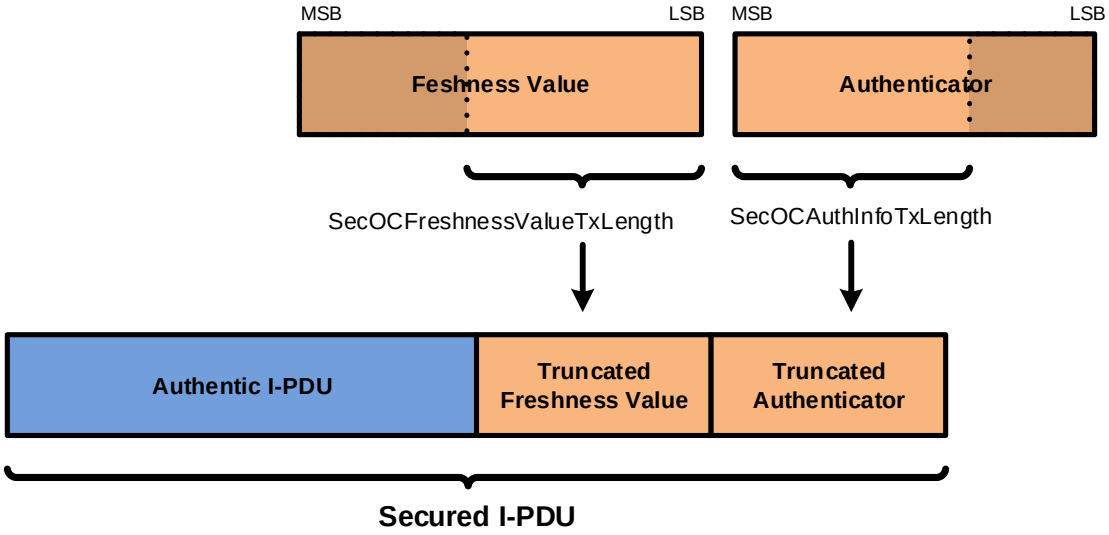

Figure 5: Secured I-PDU contents with truncated Freshness Counter and truncated Authenticator

Note: For the resource constraint embedded use case with static participants, we propose using Message Authentication Codes (MACs) as a basis for authentication (e.g. a CMAC [16] based on AES [19] with an adequate key length).

Note: In case a MAC is used, it is possible to transmit and compare only parts of the MAC. This is known as MAC truncation. However, this results in a lower security level at least for forgery of single MACs. While we propose to always use a key length of at least 128 bit, a MAC truncation can be beneficial. Of course, the actual length of the MAC for each use case has to be chosen carefully. For some guidance, we refer to appendix A of [16]. In general, MAC sizes of 64 bit and above are considered to provide sufficient protection against guessing attacks by NIST. Depending on the use case, different MAC sizes can be appropriate, but this requires careful judgment by a security expert.

[SWS_SecOC_00011]⌈
All SecOC data (i.e. Freshness Value, Authenticator, Data Identifier) that is directly or indirectly transmitted to the other side of a communication link shall be encoded in Big Endian byte order so that each SecOC module interprets the data in the same way.
⌋ (SRS_SecOC_00006)

7.1.1.2 Data covered by Authenticator

The data that the Authenticator is calculated on consists of the Data Identifier of the Secured I-PDU (parameter SecOCDataId), Authentic I-PDU data, and the Complete Freshness Value. The Data Identifier of the Secured I-PDU (parameter SecOCDataId), the complete Authentic I-PDU, and the complete Freshness Value are concatenated together respectively to make up the bit array that is passed into the authentication algorithm for Authenticator generation/verification.

DataToAuthenticator = Data Identifier | Authentic I-PDU | Complete Freshness Value

Note: “|” denotes concatenation

7.1.1.3 Freshness Counters and Freshness Timestamps

Each Secured I-PDU is configured with at least one Freshness Value. The Freshness Value refers to a monotonic counter that is used to ensure freshness of the Secured I-PDU. Such a monotonic counter could be realized by means of individual message counters, called Freshness Counter, or by a time stamp value called Freshness Timestamp.

[SWS_SecOC_00015]⌈
If SecOCUseFreshnessTimestamp is set to TRUE, the SecOC module shall use a Freshness Timestamp to generate the Freshness Value.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

Note: As base for the Freshness Timestamp, the global synchronized time can be used. As this global synchronized time will have the same value at the sender and all receivers, its value can be used as Freshness Value with the advantage that it does not necessarily need to be transmitted within the Secured PDU itself and it does not need to be transmitted for every sender and receiver individually.

[SWS_SecOC_00091]⌈
If SecOCUseFreshnessTimestamp is set to TRUE, the parameter SecOCFreshnessTimestampTimePeriodFactor shall be used to configure the resolution.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

Note: The parameter specifies a multiplication factor that defines the actual resolution of the timestamp with a basis in microseconds. It is chosen such that transmission delays and jitters are compensated. Moreover, the resolution of the timestamp should consider the maximum expected deviation between the global time master and the receiving nodes.

Note: e.g. the global synchronized time may provide a resolution of one microsecond and the maximum expected deviation of the global synchronized time value is one hundred microseconds whereas for securing against replay attacks an accuracy of several milliseconds would be sufficient. In this case, the resolution of the timestamp could be reduced to one millisecond.

[SWS_SecOC_00092]⌈
If SecOCUseFreshnessTimestamp is set to TRUE, an acceptance window shall be defined for each receiver of a secured I-PDU using SecOCRxAcceptanceWindow.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

Note: E.g. the acceptance window for a receiver could be configured to one second, meaning that the timestamp value on the sender side is allowed to be one second more or one second less than the current value on the receiver's side. In this case, the receiver would first try to authenticate the received message using its current timestamp value. If this authentication fails it would try the authentication again with a timestamp value incremented / decremented as long as the timestamp value is not larger/lower than the timestamp value +/- the value of SecOCRxAcceptanceWindow w.r.t the resolution of the timestamp (for details please refer to SWS_SecOC_00053).

Note: If possible, the resolution of the timestamp should be chosen such, that no acceptance window is needed, that is that the expected deviation between sender and receiver is zero w.r.t. the configured resolution of the timestamp.

[SWS_SecOC_00093]⌈
If SecOCUseFreshnessTimestamp is set to FALSE, SecOC shall use individual freshness counters to generate the freshness value. The SecOC module shall provide a Freshness Counter for each configured Freshness Value ID (parameter SecOCFreshnessValueId and SecOCSecondaryFreshnessValueId).
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

[SWS_SecOC_00094]⌈
If the parameter SecOCFreshnessValueTxLength is configured to a smaller length than the actual freshness value, SecOC shall include only the least significant bits of the freshness value up to SecOCFreshnessValueTxLength within the secured I-PDU.
If the parameter SecOCFreshnessValueTxLength is configured to 0, the freshness value shall not be included in the secured I-PDU.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)
Note: The larger number of bits of the complete Freshness Value included in the authenticated message payload results in a larger window where the receiver remains synchronized with the transmitters Freshness Value without executing a synchronization strategy.

Note: When including part of the Freshness Value in the authenticated message payload, the Freshness Value is referred to as two parts, the most significant bits and the least significant bits. The part of the counter included in the Secured I-PDU payload is referred to as the least significant bits of the Freshness Value and the remaining part of the counter is referred to as the most significant bits of the Freshness Value.

[SWS_SecOC_00017]⌈

The Freshness Value shall not roll over or overflow for the life of the Key used to generate/verify corresponding Authenticators.
] (SRS_SecOC_00007)

Note: The length of the value (parameter SecOCFreshnessValueLength) should be determined based on the expected lifetime of the corresponding key and the expected frequency of value increments.

Note: The Freshness Value is always linked to a Key. Decreasing/resetting the value (at sender and/or receiver side) is ONLY allowed during a key update/initialization process.

[SWS_SecOC_00168]⌈
The SecOC shall check if the Freshness Value has reached its limit and thus might overflow. If the SecOC detects that a Freshness Value has reached its limit, it shall stop sending or verifying Secured I-PDUs that are related to that Freshness Value.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

[SWS_SecOC_00018]⌈
Upon update/initialization of a new key when Freshness Counters are used, the sender counter shall be set to 1 and the receiver counter to 0.
⌋ (SRS_SecOC_00007)

[SWS_SecOC_00019]⌈
As long as the key has not changed, the Freshness Counter shall be set to the last know valid counter value stored in NVM.
⌋ (SRS_SecOC_00020, SRS_SecOC_00007)

To properly ensure freshness, the Freshness Value on both sides of the communication channel should be incremented synchronically.

[SWS_SecOC_00020]⌈
The Freshness Counter has to be incremented for each outgoing message that is intended to be recognized as an individual incoming message on the receiver side. On the receiver side, the MAC verification of each received message including the counter update shall be performed exactly once.
⌋ (SRS_SecOC_00007)

[SWS_SecOC_00021]⌈
If verification of the Secured I-PDU fails and either SecOCFreshnessCounterSyncAttempts or SecOCRxAcceptanceWindow is configured to a value greater than 0, the SecOC module shall reevaluate the Secured I-PDU using a different Freshness Value before considering the received data as non-authentic (e.g. counter or time de-synchronization is suspected to be the reason of the failed authentication verification).
⌋ (SRS_SecOC_00007)

[SWS_SecOC_00022]⌈
The number of verification attempts using a different Freshness Value before considering the received data as non-authentic shall be limited by SecOCFreshnessCounterSyncAttempts in case of SecOCUseFreshnessTimestamp is set to FALSE or by SecOCRxAcceptanceWindow in case of

SecOCUseFreshnessTimestamp is set to TRUE.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

7.1.1.4 Secondary Freshness Value

When a Secondary Freshness Value is configured, the Freshness Value previously described in this document is referred to as the Primary Freshness Value.

[SWS_SecOC_00023]⌈
If a Secured I-PDU configured with a Secondary Freshness Value, the length of the Secondary Freshness Value shall have the same length as the corresponding Primary Freshness Value for that I-PDU.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

[SWS_SecOC_00024]⌈
If a Secondary Freshness Value is configured for a Secured I-PDU and the authentication verification fails for that PDU using the counter value corresponding to the Primary Freshness Value, authentication verification shall be re-attempted using the value corresponding to the Secondary Freshness Value.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

Note: The Secondary Freshness Value is only applicable to the receiving ECU and thus is not transmitted nor used to generate the Authenticator.

[SWS_SecOC_00028]⌈
Regardless if a Secondary Freshness Counter is configured for a Secured I-PDU, the counter value corresponding to the Primary Freshness Value shall always be used first to attempt authentication verification.
⌋ (SRS_SecOC_00007)

Support and usage of a Secondary Freshness Value is optional. If a Secondary Value is used, this adds flexibility to the counter synchronization strategies.

In case the counter value corresponding to the Primary Freshness Value fails authentication verification and the counter value corresponding to Secondary Freshness Value results in successful authentication verification, OEM specific software should utilize the SecOC_FreshnessValueRead and SecOC_FreshnessValueWrite interfaces to replace the counter value corresponding to the Primary Freshness Value with the counter value corresponding to Secondary Freshness Value.

Note: This will ensure that during future authentication attempts, the first counter value considered is the counter value that most recently resulted in successful authentication verification.

In order to not influence security negatively, some general (monotony) rules for usage of multiple values should be considered:

- Secondary value should never be smaller than primary counter.
- Primary value has to stay monotonous, also when updated according to successful verification using secondary counter.

Usage Example:

A Secured I-PDU on receiver side is configured with a Freshness Value ID and a Secondary Freshness Value ID. An OEM specific SWC managing the counter values suspects the counter shared between the transmitter and receiver are out of sync. Based on an OEM specific algorithm, the SWC managing the counters determines a new counter value that should be used if indeed the counters are out of sync. The OEM SWC managing the counters then overwrites the counter corresponding to Secondary Freshness Value ID with the new predetermined counter value using the SecOC_FreshnessValueWrite interface.

A Secured I-PDU is received and authentication verification is initially performed based on the counter value corresponding to Freshness Value ID. Assume the counter value corresponding to Freshness Value ID resulted in failed authentication verification. At this point SecOC does not explicitly inform the consuming SWC of the failed authentication status, rather it re-evaluates authentication using the counter value corresponding to Secondary Freshness Value ID. If the counter value corresponding to Secondary Freshness Value ID results in successful authentication verification, SecOC sends the received data to the consuming SWC. The OEM specific SWC managing the counters is informed of the status for each verification attempt by means of the SecOC_VerifyStatus interface (see Section 8.7.2.1). In case of a successful verification using the Secondary Freshness Value, the OEM specific SWC overwrites the counter value corresponding to Freshness Value ID with the counter value corresponding to Secondary Freshness Value ID via the SecOC_FreshnessValueRead and SecOC_FreshnessValueWrite interfaces.

Note: The OEM specific SWC managing the counters utilize the SecOC_VerificationStatus interface to determine which counts passed/failed authentication verification.

7.1.2 Authentication of I-PDUs

[SWS_SecOC_00031]⌈
The creation of a Secured I-PDU and thus the authentication of an Authentic I-PDU consists of the following six steps:

1. Prepare Secured I-PDU
2. Construct Data to Authenticator
3. Generate Authenticator
4. Construct Secured I-PDU
5. Increment Freshness Counter
6. Broadcast Secured I-PDU

⌋ (SRS_SecOC_00006)

[SWS_SecOC_00033]⌈
The SecOC module shall prepare the Secured I-PDU. During preparation, SecOC shall allocate the necessary buffers to hold the intermediate and final results of the authentication process.
⌋ (SRS_SecOC_00006)

[SWS_SecOC_00034]⌈
The SecOC module shall construct the `DataToAuthenticator`, i.e. the data that is used to calculate the Authenticator. `DataToAuthenticator` is formed by

concatenating the full 16 bit representation of the Data Id (parameter SecOCDataId), the complete Authentic I-PDU and the complete Freshness Value corresponding to SecOCFreshnessValueID in the given order. The Data Id and the Freshness Value shall be encoded in Big Endian byte order for that purpose.
⌋ (SRS_SecOC_00006)

[SWS_SecOC_00035]⌈
The SecOC module shall generate the Authenticator by passing DataToAuthenticator, length of DataToAuthenticator and a pointer to a key corresponding to SecOCKeyID into the Authentication Algorithm corresponding to SecOCTxAuthServiceConfigRef.
⌋ (SRS_SecOC_00006)

[SWS_SecOC_00036]⌈
The SecOC module shall truncate the resulting Authenticator down to the number of bits specified by SecOCAuthInfoTxLength.
⌋ (SRS_SecOC_00006)

[SWS_SecOC_00037]⌈
The SecOC module shall construct the Secured I-PDU by adding the Freshness Value and the Authenticator to the Authentic I-PDU.
⌋ (SRS_SecOC_00006)

Note: The Freshness Counter and the Authenticator included as part of the Secured I-PDU may be truncated per configuration specific to the identifier of the Secured I-PDU. The scheme for the Secured I-PDU looks as follows:

```
SecuredPDU =
AuthenticIPDU
| FreshnessValue [SecOCFreshnessValueTxLength]
| Authenticator [SecOCAuthInfoTxLength]
```

[SWS_SecOC_00038]⌈
If SecOCUseFreshnessTimestamp is set to FALSE, the SecOC module shall increment the Freshness Counter corresponding to SecOCFreshnessValueID by 1 (CNT ++) only if it has started the transmission of the Secured I-PDU by calling the PduR for further routing.
⌋ (SRS_SecOC_00006)

Note: If the transmission of the Secured I-PDU has been cancelled before, it should not increment the Freshness Counter corresponding to SecOCFreshnessValueID.

7.1.3 Verification of I-PDUs

[SWS_SecOC_00040]⌈
The verification of a Secured I-PDU consists of the following six steps:

- Parse Authentic I-PDU, Freshness Value and Authenticator
- Construct Freshness Value
- Construct Data to Authentication
- Verify Authentication Information
- Set Freshness Value

- Pass Authentic I-PDU to upper layer

⌋ (SRS_SecOC_00006)

[SWS_SecOC_00042]⌈
Upon reception of a secured I-PDU, SecOC shall parse the Authentic I-PDU, the Freshness Value and the Authenticator from it.
⌋ (SRS_SecOC_00006)

[SWS_SecOC_00045]⌈
If SecOCUseFreshnessTimestamp is set to FALSE, the SecOC module shall construct Freshness Verify Value (i.e. the Freshness Value to be used for Verification). In the event the complete Freshness Value is transmitted in the secured I-PDU, it needs to be verified that the constructed FreshnessVerifyValue is larger than the last stored notion of the Freshness Value. If it is not larger than the last stored notion of the Freshness Value, the SecOC module shall stop the verification and drop the Secured I-PDU.

Otherwise, constructing the Authentication Verify Counter is defined as outlined by the following pseudo code.

```
If (SecOCFreshnessValueTxLength = FreshnessValueLength)
{
   FreshnessVerifyValue = FreshnessValue parsed from Secured I-PDU;
}
Else
{
   If (FreshnessValue parsed from Secured I-PDU > least significant bits of FreshnessValue corresponding to SecOCFreshnessValueID)
   {
      Attempts = 0;
      FreshnessVerifyValue =
      most significant bits of FreshnessValue corresponding to SecOCFreshnessValueID | FreshnessValue parsed from Secured I-PDU;
   }
   Else
   {
      Attempts = 0;
      FreshnessVerifyValue =
      most significant bits of FreshnessValue corresponding to SecOCFreshnessValueID + 1 | FreshnessValue parsed from payload;
   }
}
```

⌋ (SRS_SecOC_00002, SRS_SecOC_00007, SRS_SecOC_00022)

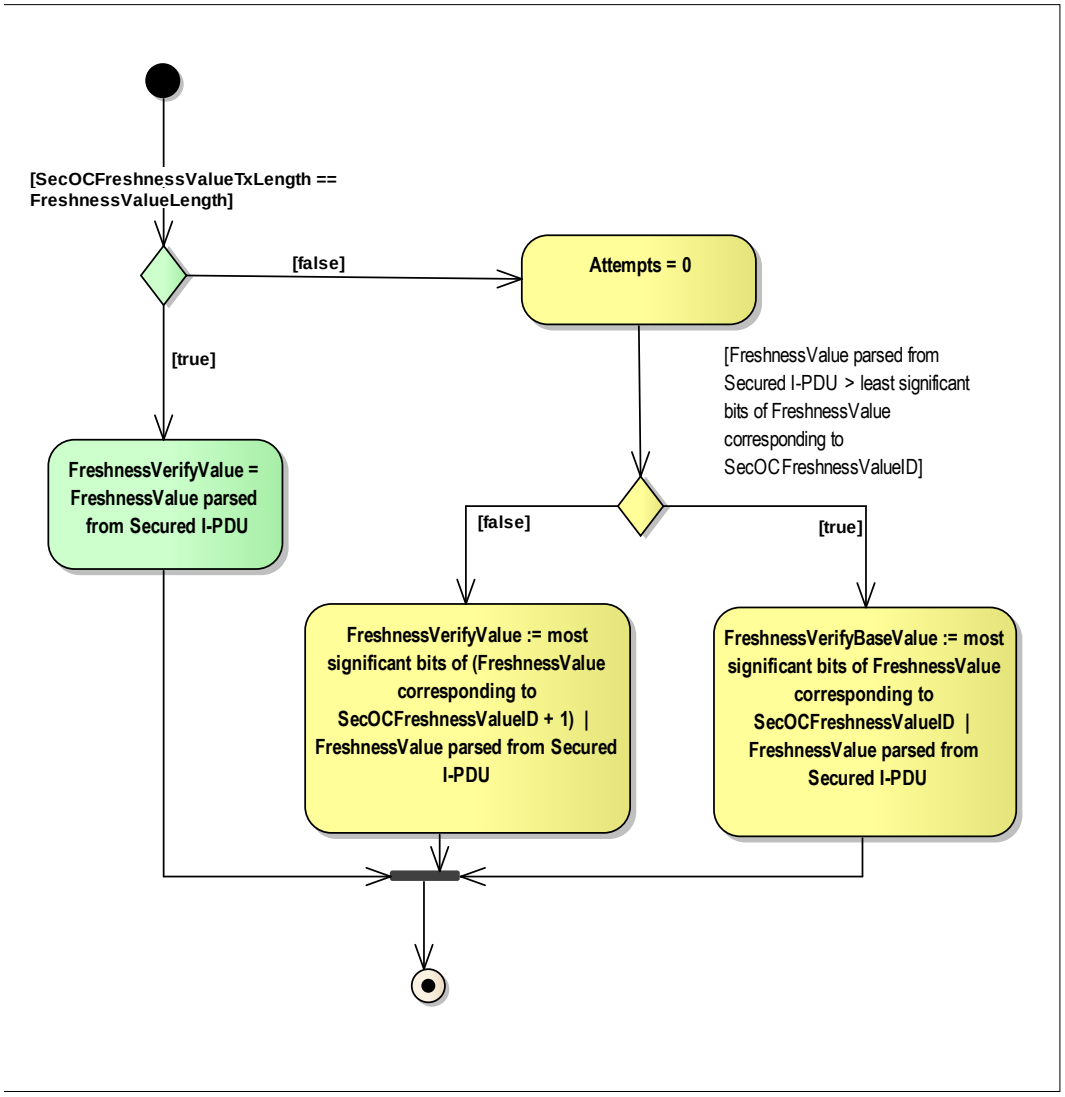

Figure 6: Construction of Freshness Value

[SWS_SecOC_00052]⌈
If SecOCUseFreshnessTimestamp is set to TRUE, the SecOC module shall construct Freshness Verify Value (i.e. the Freshness Value to be used for Verification). In case of complete Freshness Value transmission, it needs to be verified that the constructed *FreshnessVerifyValue* is within the acceptance window defined by *SecOCRxAcceptanceWindow*. If it is not in that window, the SecOC module shall stop the verification and drop the Secured I-PDU.

Otherwise, constructing the Authentication Verify Value is defined as outlined by the following pseudo code.

```
If (SecOCFreshnessValueTxLength = FreshnessValueLength)
{
   FreshnessVerifyValue = FreshnessValue parsed from Secured I-PDU;
}
Else
{
   If ((most significant bits of FreshnessValue corresponding to SecOCFreshnessValueID | FreshnessValue parsed from Secured I-PDU)
      < (max(0: (most significant bits of FreshnessValue corresponding to SecOCFreshnessValueID | least significant bits of
      FreshnessValue corresponding to SecOCFreshnessValueID) - SecOCRxAcceptanceWindow)))
   {
      Attempts = 0;
      FreshnessVerifyBaseValue = most significant bits of FreshnessValue corresponding to SecOCFreshnessValueID + 1;
   }
   Else
```

```
   {
      Attempts = 0;
      FreshnessVerifyBaseValue = most significant bits of FreshnessValue corresponding to SecOCFreshnessValueID;
   }
   FreshnessVerifyValue = FreshnessVerifyUpperValue = FreshnessVerifyLowerValue =
   FreshnessVerifyBaseValue | FreshnessValue parsed from Secured I-PDU;
}
```

] (SRS_SecOC_00002, SRS_SecOC_00007, SRS_SecOC_00022)

[SWS_SecOC_00046][
The SecOC module shall construct the data that is used to calculate the Authenticator (DataToAuthenticator) on the receiver side. This data is comprised of `SecOCDataId | AuthenticIPDU | FreshnessVerifyValue`
] (SRS_SecOC_00006)

[SWS_SecOC_00047][
If SecOCUseFreshnessTimestamp is set to FALSE, the SecOC module shall verify the Authenticator by passing `DataToAuthenticator`, length of `DataToAuthenticator`, handle to a key corresponding to `SecOCKeyID`, the `Authenticator` parsed from Secured I-PDU, and `SecOCAuthInfoTxLength` into the authentication algorithm corresponding to `SecOCRxAuthServiceConfigRef`. The verification process is repeated as outlined in the following pseudo code:

```
If (Authentication Verification Passes)
{
   Pass Authentic I-PDU to upper layer;
}
If (Authentication Verification Fails && Attempts < SecOCFreshnessCounterSyncAttempts)
{
   Attempts ++;
   Increment most significant bits of FreshnessVerifyValue by 1;
   Re-attempt Authentication;
}
Else
{
   Drop message;
}
```

] (SRS_SecOC_00002, SRS_SecOC_00007, SRS_SecOC_00022)

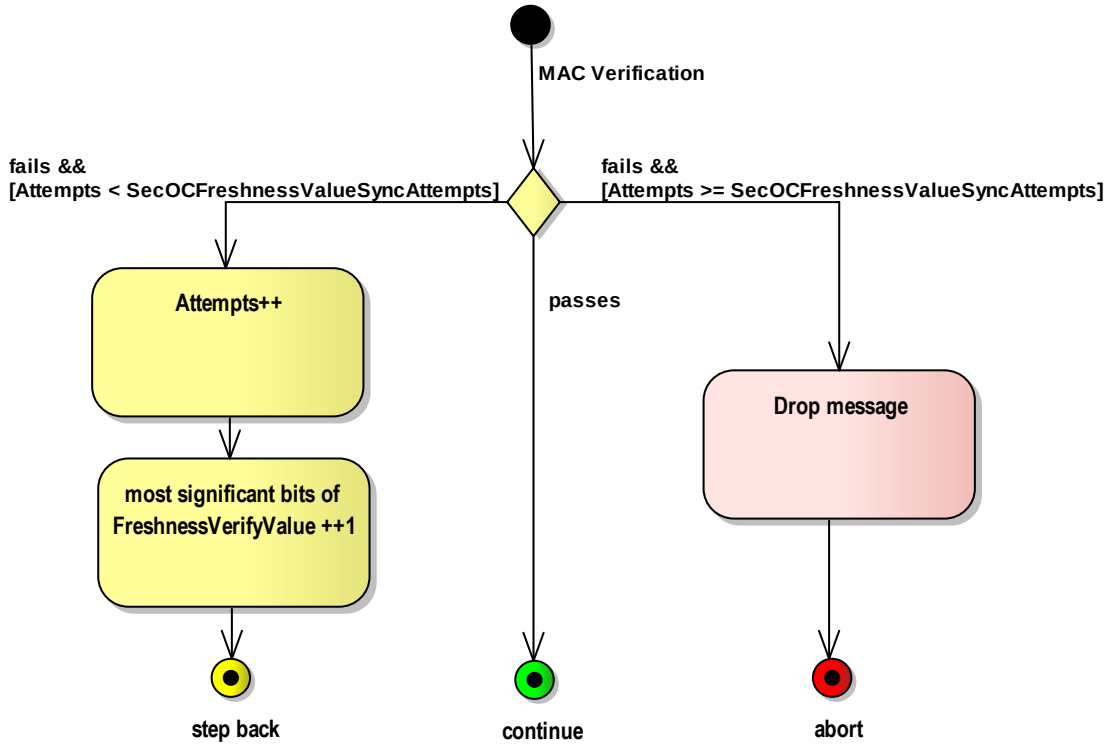

Figure 7: Verification of MAC

[SWS_SecOC_00053]⌈
If SecOCUseFreshnessTimestamp is set to TRUE, the SecOC module shall verify the Authenticator by passing `DataToAuthenticator`, length of `DataToAuthenticator`, handle to a key corresponding to `SecOCKeyID`, the `Authenticator` parsed from Secured I-PDU, and `SecOCAuthInfoTxLength` into the authentication algorithm corresponding to `SecOCRxAuthServiceConfigRef`. The verification process is repeated as outlined in the following pseudo code:

```
If (Authentication Verification Passes)
{
   Pass Authentic I-PDU to upper layer;
}
Else
{
   Attempts ++;
   If (Attempts mod 2! == 0)
   {
      FreshnessVerifyValue = FreshnessVerifyLowerValue =
      (FreshnessVerifyBaseValue - floor(Attempts/2)+1) | FreshnessValue parsed from Secured I-PDU;
      // check lower bound
      If ((FreshnessValue corresponding to SecOCFreshnessValueID – SecOCRxAcceptanceWindow) <= FreshnessVerifyLowerValue)
      {
         Re-attempt Authentication;
      }
   }
   Else
   {
      FreshnessVerifyValue = FreshnessVerifyUpperValue =
      (FreshnessVerifyBaseValue + (Attempts/2)) | FreshnessValue parsed from Secured I-PDU;
      // check upper bound
      If ((FreshnessValue corresponding to SecOCFreshnessValueID + SecOCRxAcceptanceWindow) >= FreshnessVerifyUpperValue)
      {
         Re-attempt Authentication;
      }
   }
   // check upper and lower bound
   If (((FreshnessValue corresponding to SecOCFreshnessValueID – SecOCRxAcceptanceWindow) > FreshnessVerifyLowerValue)
      && ((FreshnessValue corresponding to SecOCFreshnessValueID + SecOCRxAcceptanceWindow) < FreshnessVerifyUpperValue))
```

```
  {
     Drop message;
  }
}
```

⌋ (SRS_SecOC_00002, SRS_SecOC_00007, SRS_SecOC_00022)

[SWS_SecOC_00048]⌈
The SecOC module shall report each individual verification status (the final one as well as all intermediate ones) by serving the call out function SecOC_VerificationStatusCallout and the SecOC_VerificationStatus interface according to its current configuration (see parameter SecOCVerificationStatusPropagationMode).
⌋ (SRS_SecOC_00022)

7.1.3.1 Successful verification of I-PDUs

[SWS_SecOC_00049]⌈
If the verification of a Secured I-PDU was successful and If SecOCUseFreshnessTimestamp is set to FALSE, the SecOC module shall set the Freshness Value corresponding to the successfully used Freshness Value (i.e. `SecOCFreshnessValueID` or `SecOCSecondaryFreshnessValueID)` equal to `FreshnessVerifyValue`.
⌋ (SRS_SecOC_00002, SRS_SecOC_00007)

[SWS_SecOC_00050]⌈
Only if the verification of a Secured I-PDU was successful, the SecOC module shall pass the `Authentic I-PDU` to the upper layer communication modules using the lower layer interfaces of the PduR.
⌋ (SRS_SecOC_00022)

Note: In case the verification has eventually failed, the SecOC module must not pass the Authentic I-PDU to the PduR for further routing.

7.1.4 Adaptation in case of asymmetric approach

Although this document consequently uses the terms and concepts from synchronous cryptography, the SecOC module can be configured to use both, synchronous as well as asynchronous cryptographic algorithms. In case of an asymmetric approach using digital signatures instead of the MAC-approach described throughout the whole document, some adaptations have to be made:

1. Instead of a shared secret between sender and (all) receivers, a key pair consisting of public key and secret key is used. The secret (or private) key is used by the sender to generate the signature, the corresponding public keys is used by (all) receiver(s) to verify the signature. The private key must not be feasibly computable from the public key and it shall not be assessable by the receivers.
2. In order to verify a message, the receiver needs access to the complete signature /output of the signature generation algorithm. Therefore, a truncation of the signature as proposed in the MAC case is NOT possible. The parameter SecOCAuthInfoTxLength has to be set to the complete length of the signature.

3. The signature verification uses a different algorithm then the signature generation. So instead of „rebuilding“ the MAC on receiver side and comparing it with the received (truncated) MAC as given above, the receiver / verifier performs the verification algorithm using the DataToAuthenticator (including full counter) and the signature as inputs and getting a Boolean value as output, determining whether the verification passed or failed.

7.2 Relationship to PduR

The SecOC module is arranged next to the PDU-Router in the layered architecture of AUTOSAR; see Figure 8.

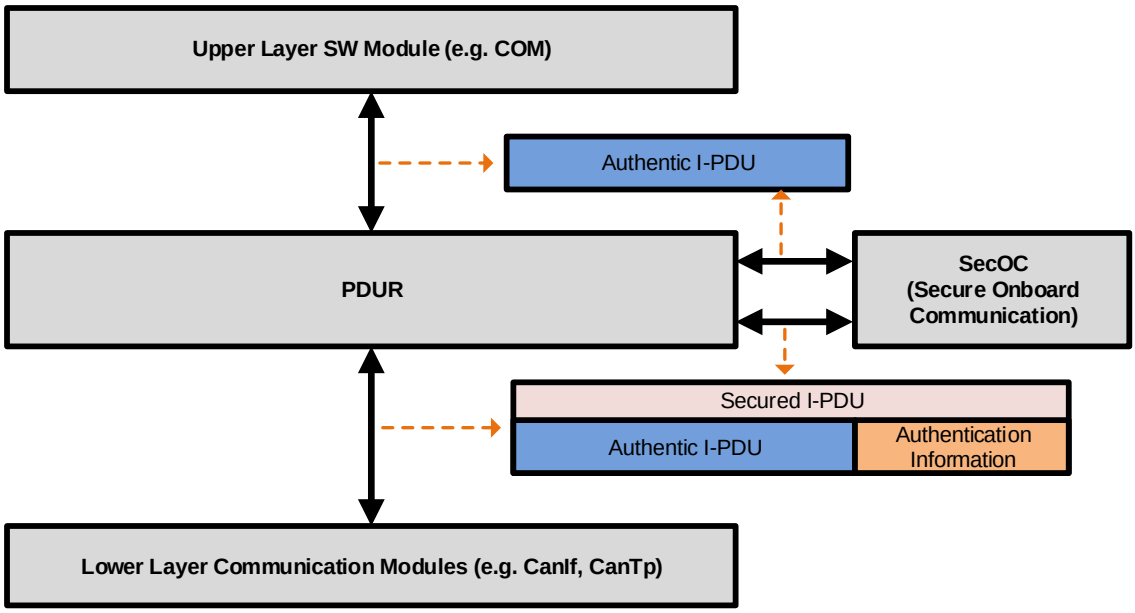

Figure 8 Transformation of an Authentic I-PDU in a Secured I-PDU by SecOC

[SWS_SecOC_00153]⌈
The SecOC module shall be implemented so that no other modules depend on it and that it is possible to build a system without the SecOC module if it is not needed.
⌋ (SRS_BSW_00171)

7.3 Initialization

The SecOC module provides an initialization function (SecOC_Init) as defined in SWS_SecOC_00106. This function initializes all internal global variables and the buffers to store the SecOC I-PDUs and all intermediate results. The environment of the SecOC shall call SecOC_Init before calling any other function of the SecOC module except SecOC_GetVersionInfo. The implementer has to ensure that SecOC_E_UNINIT is returned in development mode in case an API function is called before the module is initialized.

For the I-PDU data transmission pathway through the SecOC module, a buffer is allocated inside the SecOC module. This buffer needs to be initialized because it might be transmitted before it has been fully populated with data by the upper layer of lower layer communication modules.

[SWS_SecOC_00054]⌈
Within SecOC_Init, the module shall initialize all internal global variables and the buffers of the SecOC I-PDUs.
⌋ (SRS_SecOC_00005)

[SWS_SecOC_00055]⌈
Within SecOC_Init, the module shall restore all Freshness Value values from NVRAM so that all counter values show the status they had before the SecOC has been shut down.
⌋ (SRS_SecOC_00020)

7.4 Authentication of outgoing PDUs

The term authentication describes the creation of a Secured I-PDU by adding Authentication Information to an Authentic I-PDU. This process is described in general terms in Section 7.1.2. This section refines the general description with respect to requirements arising from the integration with the PduR module considering different bus interfaces and transport protocols. In general, the interaction with the PduR module and the authentication of Authentic I-PDUs are organized according to the following scheme:

1. For each transmission request of an Authentic I-PDU, the upper layer communication module shall call the PduR module through PduR_<Up>Transmit.
2. The PduR routes this request to the SecOC module and calls SecOC_Transmit.
3. The SecOC module copies the Authentic I-PDU to its own memory and returns.
4. During the next scheduled call of its main function, the SecOC module creates the Secured I-PDU by calculating the Authentication Information and initiates the transmission of the Secured I-PDU by notifying the respective lower layer module via the PduR module.
5. Thereafter, the SecOC module takes the role of an upper layer communication module and thus serves all lower layer requests to provide information on or to copy data of the Secured I-PDU.
6. Finally, the confirmation of the successful or unsuccessful transmission of the Secured I-PDU are provided to the upper layer communication module as confirmation of the successful or unsuccessful transmission of the Authentic I-PDU

Note: For each Authentic I-PDU, the upper layer communication module shall be configured in such a way that it calls the PduR module as it normally does for a direct transmission request. In this case, the upper layer is decoupled from TriggerTransmit and TP behavior by means of the SecOC module.

The SecOC module decouples the interaction between upper layer modules and lower layer modules. It gets all transmission relevant information to be transmitted and thus could manage the interaction with lower layer module on its own and without affecting the upper layer module.

To initiate the transmission of an Authentic I-PDU, the upper layer module always (and independent of the bus interface that is used for the concrete transmission) calls the PduR module through PduR_<Up>Transmit. The PduR routes this request to the SecOC module so that the SecOC module has immediate access to the Authentic I-PDU in the buffer of the upper layer communication module.

[SWS_SecOC_00057]⌈
The SecOC module shall provide sufficient buffer capacities to store the incoming Authentic I-PDU, the outgoing Secured I-PDU and all intermediate data of the authentication process according to the process described in SWS_SecOC_00031.
⌋ (SRS_SecOC_00006)

[SWS_SecOC_00146]⌈
The SecOC module shall provide separate buffers for the Authentic I-PDU and the Secured I-PDU.
⌋ (SRS_SecOC_00006)

[SWS_SecOC_00110]⌈
Any transmission request from the upper layer communication module shall overwrite the buffer that contains the Authentic I-PDU without affecting the buffer of the respective Secured I-PDU.
⌋ (SRS_BSW_00426)

Thus, upper layer updates for Authentic I-PDUs could be processed without affecting ongoing transmission activities of Secured I-PDUs with the lower layer communication module.

7.4.1 Authentication during direct transmission

For transmission of an Authentic I-PDU using bus interfaces that allow ad-hoc transmission (e.g. CanIf), the PDU Router module triggers the transmit operation of the SecOC module for an Authentic I-PDU. In this case, the SecOC module prepares the creation of a Secured I-PDU on basis of the Authentic I-PDU by allocating internal buffer capacities and by copying the Authentic I-PDU to a local buffer location. Afterwards it returns SecOC_Transmit.

[SWS_SecOC_00058]⌈
The SecOC module shall allocate internal buffer capacities to store the Authentic I-PDU and the Authentication Information in a consecutive memory location.
⌋ (SRS_SecOC_00006)

The actual creation of the Secured I-PDU is processed during the next subsequent call of the scheduled main function. This includes calculating the Authentication Information according to SWS_SecOC_00031 and adding the Authentication Information (i.e. the Authenticator and the possibly truncated Freshness Value) consecutively to the buffer location directly behind the Authentic I-PDU. Thereafter, SecOC module triggers the transmission of the Secured I-PDU to the destination lower layer module by calling PduR_SecOCTransmit at the PduR.

[SWS_SecOC_00060]⌈
For transmission of Authentic I-PDUs using bus interfaces that allow ad-hoc transmission (e.g. CanIf), the SecOC module shall calculate the Authenticator in the scheduled main function according to the overall approach specified in SWS_SecOC_00031.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00061]⌈
For transmission of Authentic I-PDUs using bus interfaces that allow ad-hoc communication (e.g. CanIf), the SecOC module shall create the Secured I-PDU in the scheduled main function.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00062]⌈
The SecOC module shall provide the complete Secured I-PDU for further transmission to the destination lower layer module by triggering PduR_SecOCTransmit.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00063]⌈
If the PDU Router module notifies the SecOC module that the destination lower layer module has confirmed the transmission of the Secured I-PDU by calling SecOC_TxConfirmation, the SecOC module shall confirm the reception of the respective Authentic I-PDU to the upper layer module by calling PduR_SecOCTxConfirmation.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00064]⌈
For transmission of Authentic I-PDUs using bus interfaces that allow ad-hoc communication (e.g. CanIf), the SecOC module shall free the buffer that contains the Secured I-PDU if SecOC_TxConfirmation is called for the Secured I-PDU.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

7.4.2 Authentication during triggered transmission

For transmission of an Authentic I-PDU using bus interfaces that allow triggered transmission (e.g. FrIf), the upper layer is configured in such a way that it calls the PduR module like it normally does for a direct transmission (see SWS_SecOC0054). Thus, the upper layer module immediately provides access to the Authentic I-PDU by providing the required buffer information through PduR_<Up>Transmit. The PduR forwards this transmission request to the SecOC module by calling SecOC_TriggerTransmit.

Note: Authentication for triggered transmission is only supported, if the upper layer initiates the transmission by explicitly calling PduR_<Up>Transmit in before. Triggered transmission in mode AlwaysTransmit shall not be used.

In turn, the SecOC module allocates sufficient buffer capacities to store the Authentic I-PDU, the Secured I-PDU and all intermediate data of the authentication process. The SecOC module copies the Authentic I-PDU into its own buffer and returns (see SWS_SecOC_00057, SWS_SecOC_00058, SWS_SecOC_00059).

The actual creation of the Secured I-PDU is processed during the subsequent call of the scheduled main function. This includes calculating the Authentication Information according to SWS_SecOC_00031 and adding the Authentication Information (i.e. the Authenticator and the possibly truncated Freshness Value) consecutively to the buffer location directly behind the Authentic I-PDU. Thereafter, SecOC module triggers the transmission of the Secured I-PDU to the destination lower layer module by calling PduR_SecOCTransmit at the PduR.

[SWS_SecOC_00065]⌈
For transmission of Authentic I-PDUs using bus interfaces that allow triggered transmission (e.g. FrIf), the SecOC module shall calculate the Authenticator in the scheduled main function according to the overall approach specified in SWS_SecOC_00031.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00066]⌈
For transmission of Authentic I-PDUs using bus interfaces that allow triggered transmission (e.g. FrIf), the SecOC module shall create the Secured I-PDU in the scheduled main function.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

In the following, the SecOC module serves as a data provider for the subsequent transmission request from the lower layer module. Thus, the SecOC module holds the complete Secured I-PDU and acts as the upper layer module. The upper layer module does not expect any further call back that request the copying of the Authentic I-PDU to the lower layer module.

[SWS_SecOC_00067]⌈
For transmission of Authentic I-PDUs using bus interfaces that allow triggered transmission (e.g. FrIf), the SecOC module shall indicate the transmission request for the complete Secured I-PDU by triggering PduR_SecOCTransmit at the PduR. The PduR is responsible to further process the request and to notify the respective lower layer module.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

The destination lower layer module calls PduR_<Lo>TriggerTransmit when it is ready to transmit the Secured I-PDU. PduR forwards this request to the SecOC module and the SecOC module copies the complete Secured I-PDU to the lower layer. Afterwards it returns.

Note: The SecOc module must not forward the trigger transmit call to the upper layer but takes itself the role of the upper layer and copies the complete Secured I-PDU to the lower layer.

[SWS_SecOC_00068]⌈
When SecOC_TriggerTransmit is called by the PduR module, the SecOC module shall copy the Secured I-PDU to the lower layer destination module.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00150]⌈
When SecOC_TriggerTransmit is called by the PduR module and the SecOC module is not able to provide a Secured I-PDU to the lower layer (no Secured I-PDU available), the SecOC module shall return the call with E_NOT_OK.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

Finally, when the lower layer confirms the transmission of the Secured I-PDU via PduR_<Lo>TxConfirmation, the confirmation is forwarded to the SecOC module by calling SecOC_TxConfirmation. In turn, the SecOC module confirms the transmission of the Authentic I-PDU at the PduR module so that the PduR module could forward the confirmation via <Up>_TxConfirmation to the destination upper layer module (see SWS_SecOC_00063).

During triggered transmission, the update rates of the upper layer modules and the lower layer modules might be different. Thus, the lower layer module might request a new transmission of a Secured I-PDU while the upper layer has not updated the Authentic I-PDU. In this case, the SecOC module supports the repeated transmission of the Authentic I-PDU by means of an updated Secure I-PDU. Thus, it has to preserve the Authentic I-PDU until the Secured I-PDU has been sent and its transmission has been confirmed by a means of SecOC_TxConfirmation. In this case, the SecOC module treats the existing Authentic I-PDU as new and re-authenticates it during the subsequent call to the SecOC_MainFunction.

[SWS_SecOC_00069]⌈
For transmission of Authentic I-PDUs using bus interfaces that allow triggered transmission (e.g. FrIf) and after having successfully sent the Secured I-PDU, the SecOC module shall free the buffer that contain Authentication Information and preserve the buffer that contain the Authentic I-PDU. The Authentic I-PDU shall be treated as if it has been set by the upper layer and thus shall undergo a new authentication procedure with the subsequent call of the SecOC_MainFunction.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

7.4.3 Authentication during transport protocol transmission

For transmission of an Authentic I-PDU using transport protocol transmission (e.g. CanTP, FrTp), the PDU Router module triggers the transmit operation of the SecOC module for an Authentic I-PDU. In this case, the SecOC module prepares the creation of a Secured I-PDU on basis of the Authentic I-PDU by allocation internal

buffer capacities and by copying the Authentic I-PDU to a local buffer location. Afterwards it returns SecOC_Transmit.

The actual creation of the Secured I-PDU is processed during the next following call of the scheduled main function. This includes calculating the Authentication Information according to SWS_SecOC_00031 and adding the Authentication Information (i.e. the Authenticator and the possibly truncated Freshness Value) consecutively to the buffer location directly behind the Authentic I-PDU.

Note: The overall approach of using IF API towards the upper layer and TP API towards the lower layer will not work with DCM/J1939DCM.

[SWS_SecOC_00070]⌈
For transmission of Authentic I-PDUs using transport protocol, the SecOC module shall calculate the Authenticator in the scheduled main function according to the overall approach specified in SWS_SecOC_00031.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00071]⌈
For transmission of Authentic I-PDUs using transport protocol, the SecOC module shall create the Secured I-PDU in the scheduled main function.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

Thereafter, SecOC module triggers the transmission of the Secured I-PDU to the destination lower layer module by calling PduR_SecOCStartOfReception at the PduR. Thus, it notifies the lower level module about its transmission request for the Secured I-PDU.

[SWS_SecOC_00072]⌈
For transmission of Authentic I-PDUs using transport protocol, the SecOC module shall indicate the transmission request for the complete Secured I-PDU by triggering PduR_SecOCTransmit at the PduR. The PduR is responsible to further process the request and to notify the respective lower layer module.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

In the following, the SecOC module serves as a data provider for the subsequent transmission request from the lower layer module. Thus, the SecOC module holds the complete Secured I-PDU and acts as the upper layer module. The upper layer module does not expect any further call back that request the copying of the Authentic I-PDU to the lower layer module.

When the PduR iteratively polls the SecOC module by means of SecOC_CopyTxData to effectively transmit the Secured I-PDU to a lower layer module, the SecOC module copies the NPDUs for the Secured I-PDU to the lower layer transport protocol module.

[SWS_SecOC_00073]⌈

For transmission of Authentic I-PDUs using transport protocol, the SecOC module shall copy the NPDUs addressed by SecOC_CopyTxData into the buffer of the transport protocol module. After each copy process, it returns from SecOC_CopyTxData.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

Finally, when the lower layer confirms the transmission of the Secured I-PDU via PduR_<Lo>TxConfirmation, the confirmation is forwarded to the SecOC module and the SecOC module in turn confirms the transmission of the Authentic I-PDU, so that the PduR module could forward the confirmation via <Up>_TxConfirmation to the upper layer,

[SWS_SecOC_00074]⌈
For transmission of Authentic I-PDUs using transport protocol and when the lower layer confirms the transmission of the Secured I-PDU by SecOC_TpTxConfirmation, the SecOC module shall in turn confirm the transmission of the Authentic I-PDU by PduR_SecOCTxConfirmation.⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00075]⌈
For transmission of Authentic I-PDUs using transport protocol, the SecOC module shall free the buffer that contains the Secured I-PDU only, if SecOC_TpTxConfirmation is called for the Secured I-PDU.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

7.4.4 Error handling and cancelation of transmission

[SWS_SecOC_00076]⌈
If the upper layer module requests a cancelation of an ongoing transmission of the Authentic I-PDU by calling SecOC_CancelTransmit, the SecOC module shall immediately inform the lower layer transport protocol module to cancel the ongoing transmission of the Secured I-PDU, stop all internal actions related to the Authentic I-PDU, and free all related buffers.
⌋ (SRS_SecOC_00021)

[SWS_SecOC_00077]⌈
If the lower layer transport protocol module reports an error during transmission of a Secured I-PDU using the return value E_NOT_OK, the SecOC module shall not perform any error handling other than skipping the confirmation of the transmission request for the corresponding Authentic I-PDU to the upper layer module.
⌋ (SRS_BSW_00385)

[SWS_SecOC_00151]⌈
If the CSM module reports an error during authentication of an Authentic I-PDU (authentication attempt returns E_NOT_OK), the SecOC module shall not provide a Secured I-PDU to the lower layer. It shall keep that Authentic I-PDU (if not overwritten by an incoming Authentic I-PDU of the same type) to start the authentication with the next call of the scheduled main function until the number of

additional authentication attempts for that Authentic I-PDU has reached SecOCAuthenticationRetries.
⌋ (SRS_SecOC_00021, SRS_BSW_00385)

[SWS_SecOC_00155]⌈
If the number of additional authentication attempts for an Authentic I-PDU has reached SecOCAuthenticationRetries, the SecOC module shall remove the Authentic I-PDU from its internal buffer and shall report SECOC_E_CRYPTO_FAILURE to the DET module.
⌋ (SRS_BSW_00385)

[SWS_SecOC_00108]⌈
If the SecOC module is not able to serve any upper layer or lower layer request during transmission of an Authentic I-PDU due to an arbitrary internal error, it shall return this request with E_NOT_OK.
⌋ (SRS_BSW_00385)

7.5 Verification of incoming PDUs

The term verification describes the process of comparing the Authentication Information contained in a Secured I-PDU with the Authentication Information calculated on basis of the local Data Identifier, the local Freshness Value and the Authentic I-PDU contained in the Secured I-PDU.

The process of verifying incoming Secured I-PDUs is described in general terms in Section 7.1.3. This section refines the general description with respect to requirements arising from the integration with the PduR module considering different bus interfaces and transport protocols. The overall interaction with the PduR module and the verification of Secured I-PDUs is organized as described in the following scheme:

1. For each indication of an incoming Secured I-PDU from a lower layer bus interface or transport protocol module, the SecOC module takes the role of an upper layer communication module and thus serves all lower layer requests that are necessary to receive the complete Secured I-PDU.
2. The SecOC module copies the Secured I-PDU into its own memory.
3. Thereafter, when the complete Secured I-PDU is available and during the next scheduled call of its main function, the SecOC module verifies the contents of the Secured I-PDU according to SWS_SecOC_00040.
4. If the verification fails, the SecOC module drops the Secured I-PDU.
5. If the verification succeeds, the SecOC module takes the role of a lower layer communication module and calls PduR_SecOCRxIndication for the Authentic I-PDU.
6. The SecOC reports the verification results according to SWS_SecOC_00048.

Thus, SecOC decouples the interaction between upper layer modules and lower layer modules. The SecOC module manages the interaction with lower layer module until it has copied the complete Secured I-PDU into its own buffer. It does so without affecting the upper layer module. Thereafter, it verifies the contents of the Secured I-

PDU and, dependent on the verification results, initiates the transmission of the Authentic I-PDU to the upper layer communication module.

[SWS_SecOC_00111]⌈
Any reception process initiated from the lower layer communication module shall overwrite the buffer that contains the Secured I-PDU without affecting the buffer of the respective Authentic I-PDU.
⌋ (SRS_BSW_00426)

Thus, lower layer updates of Secured I-PDUs could be processed without affecting ongoing deliveries of an Authentic I-PDU to the upper layer communication modules.

7.5.1 Verification during bus interface reception

When a Secured I-PDU is received by means of a lower layer bus interface (e.g. CanIf, FrIf), the PduR module calls SecOC_RxIndication to inform the SecOC module for each incoming Secured I-PDU. During the processing of SecOC_RxIndication, the SecOC module copies the Authentic I-PDU to its own buffer.

Note: The overall approach of using IF API towards the upper layer and TP API towards the lower layer will not work with DCM/J1939DCM.

[SWS_SecOC_00078]⌈
During reception of a Secured I-PDU that is received by means of a lower layer bus interface and when SecOC_RxIndication has been called, the SecOC module shall copy the complete Secured I-PDU into its own buffer. Afterwards it returns from SecOC_RxIndication.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

Thereafter, the actual verification of an incoming Secured I-PDU is initiated during the next call of the scheduled main function. The SecOC module extracts the Authentic I-PDU, the Authentication Information from the Secured I-PDU. The SecOC module verifies the authenticity and freshness of the Authentic I-PDU according to SecOC_SWS_0040. If the verification is successful, the SecOC indicates the reception of the Authentic I-PDU by calling PduR_SecOCRxIndication for the Authentic I-PDU. If the verification fails, the SecOC drops the PDU and does not call PduR_SecOCRxIndication.

[SWS_SecOC_00079]⌈
During reception of a Secured I-PDU that is received by means of a lower layer bus interface, the SecOC module shall verify the Authenticator according to the overall approach specified in SWS_SecOC_00040. The verification shall be processed in the scheduled main function.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00080]⌈

During reception of a Secured I-PDU that is received by means of a lower layer bus interface and if the verification eventually succeeds, the SecOC module shall call PduR_SecOCRxIndication referencing the Authentic I-PDU that is contained in the Secured I-PDU.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

Note: if the verification eventually fails, the SecOC module does not call PduR_SecOCRxIndication for the Authentic I-PDU that is contained in the Secured I-PDU.

7.5.2 Verification during transport protocol reception

When a Secured I-PDU is received by means of a lower layer transport protocol interface (e.g. CanTp, FrTp), the PduR module calls SecOC_StartOfReception to notify the SecOC module that the reception process of the respective Secured I-PDU will start.

[SWS_SecOC_00082]⌈
During reception of a Secured I-PDU that is received by means of a lower layer transport protocol interface and when SecOC_StartOfReception is called, the SecOC module shall provide buffer capacities to store the complete Secured I-PDU.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

Note: The required buffer capacities for the Secured I-PDU could be directly derived from the length of the Secured I-PDU, which is given in the system template.

When the lower layer iteratively indicates the reception of the individual NPDUs that constitute the Secured I-PDU (i.e. when SecOC_CopyRxData is called), the SecOC module copies the NPDUs to its own buffer.

[SWS_SecOC_00083]⌈
During reception of a Secured I-PDU that is received by means of a lower layer transport protocol interface and when SecOC_CopyRxData is called, the SecOC module shall copy the NPDUs addressed by SecOC_CopyRxData into its own buffers. Finally, it returns from SecOC_CopyRxData.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

Finally, when the lower layer confirms the complete reception of the Secured I-PDU via SecOC_TpRxIndication and thus the complete Secured I-PDU is available in the buffer of the SecOC module for further processing, the SecOC module starts the verification of the Authentication Information according to Section 7.1.3 during its next scheduled call of its main function.

[SWS_SecOC_00084]⌈
During reception of a Secured I-PDU that is received by means of a lower layer transport protocol interface and when SecOC_TpRxIndication is called, the SecOC module shall returns SecOC_TpRxIndication without any further processing.

⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00085]⌈
During reception of a Secured I-PDU that is received by means of a lower layer transport protocol interface and when SecOC_TpRxIndication has been called, the SecOC module shall verify the contents of the Secured I-PDU according to the process described in Section 7.1.3.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00086]⌈
During reception of a Secured I-PDU that is received by means of a lower layer transport protocol interface and when the verification eventually succeeds, the SecOC module shall call PduR_SecOCRxIndication with references to the Authentic I-PDU contained in the Secured I-PDU.
⌋ (SRS_SecOC_00010, SRS_SecOC_00010, SRS_SecOC_00012, SRS_SecOC_00013)

[SWS_SecOC_00087]⌈
The SecOC module shall free all buffer related to a Secured I-PDU either if

1. it has passed the respective authenticated I-PDU to the PduR via PduR_SecOCRxIndication,
2. the verification of a Secured I-PDU eventually failed,
3. the transmission of a Secured I-PDU has been canceled by the upper or lower layer, or
4. the transmission of a Secured I-PDU with the same Pdu Identifier has been initiated via SecOC_StartOfReception.

⌋ (SRS_SecOC_00021, SRS_SecOC_00022)

7.5.3 Error handling and cancelation of transmission

[SWS_SecOC_00089]⌈
If the lower layer transport protocol module reports an error by returning something else than E_OK during reception of a Secured I-PDU using SecOC_TpRxIndication, the SecOC module shall drop the Secured I-PDU and free all corresponding buffers.
⌋ (SRS_BSW_00385)

[SWS_SecOC_00121]⌈
If the CSM module reports an error during verification (verification attempt returns E_NOT_OK) of a Secured I-PDU, the SecOC module shall not provide the Authentic I-PDU. It shall keep the Secured I-PDU (if not overwritten by an incoming Secured I-PDU of the same type) and start the verification with the next call of the scheduled main function.
⌋ (SRS_SecOC_00022, SRS_BSW_00385)

[SWS_SecOC_00109]⌈

If the SecOC module is not able to serve any upper layer or lower layer request during reception of A Secured I-PDU due to an arbitrary internal error, it shall return this request with E_NOT_OK.
] (SRS_BSW_00385)

7.6 Gateway functionality

The SecOC module supports authentication and verification for I-PDUs that are routed from one source bus to one or more destination busses. This allows for the realization of re-authentication gateways that can be used to realize networks with different security zones or properties. The actions necessary to support the required gateway functionality can be simply derived from the authentication and verification scenarios in Sections 7.4 and 7.5. Each authentication or verification process for a given I-PDU need to be configured separately. This functionality includes:

- authentication of outgoing I-PDUs,
- verification of incoming I-PDUs,
- re-authentication gateways, i.e. the verification of incoming I-PDUs in combination of their immediate re-authentication, when the I-PDU is routed to another lower layer module.

Note: “Gatewaying-on-the-fly” is not supported by SecOC

7.7 Development Errors

[SWS_SecOC_00101] Development Error Types
[
The following errors and exceptions shall be detectable by the SecOC module depending on its build version (development/production mode):

Type or error	*Related error code*	*Value [hex]*
An API service was called with a NULL pointer	SECOC_E_PARAM_POINTER	0x01
API service used without module initialization or PduR_Init called	SECOC_E_INVALID_REQUEST	0x02
Invalid I-PDU identifier	SECOC_E_INVALID_PDU_SDU_ID	0x03
Crypto service failed	SECOC_E_CRYPTO_FAILURE	0x04
Unable to restore Freshness Value and key information from NVRAM	SECOC_E_RESTORE_FAILURE	0x05
Freshness Value at limit	SECOC_E_FRESHNESS_VALUE_AT_LIMIT	0x06

] (SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00386)

7.8 Error detection

The detection of development errors is configurable (see Section 10.2, SecOcDevErrorDetect).

7.9 Error notification

The SecOC module checks the initialization state when one of its API functions is called, and reports the DET error SECOC_E_INVALID_REQUEST in case an API call other than SecOC_Init or SecOC_GetVersionInfo occurs. Besides this, the SecOC module performs parameter checks for all called APIs. It reports the DET error SECOC_E_PARAM_POINTER when a call provides a NULL pointer and SECOC_E_INVALID_PDU_SDU_ID when a check of a I-PDU ID fails. It reports SECOC_E_CRYPTO_FAILURE when the use of CSM function finally lead to a situation that PDUs can't be authenticated or validated and SECOC_E_RESTORE_FAILURE when the SecOC module is not able to restore Freshness Values and key related information from NVRAM during initialization and SECOC_E_FRESHNESS_VALUE_AT_LIMIT when a Freshness Value has reached its limit.

[SWS_SecOC_00102]⌈
If DET reporting is enabled via SecOCDevErrorDetect and if the SecOC module has not been initialized, all functions except SecOC_Init and SecOC_GetVersionInfo shall report the error SECOC_E_INVALID_REQUEST.
⌋ (SRS_BSW_00337, SRS_BSW_00350, SRS_BSW_00385, SRS_BSW_00450)

[SWS_SecOC_00164]⌈
If DET reporting is enabled via SecOCDevErrorDetect, the SecOC module shall check the I-PDU Id parameters of its API functions against its configuration and shall report the DET error SECOC_E_INVALID_PDU_SDU_ID when an unknown I-PDU Id is referenced by the call.
⌋ (SRS_BSW_00337, SRS_BSW_00350, SRS_BSW_00385)

[SWS_SecOC_00165]⌈
If DET reporting is enabled via SecOCDevErrorDetect, the SecOC module shall report the DET error SECOC_E_RESTORE_FAILURE when it is not able to restore the Freshness Values and key related information from the NVRAM during initialization.
⌋ (SRS_BSW_00337, SRS_BSW_00350, SRS_BSW_00385)

[SWS_SecOC_00166]⌈
If DET reporting is enabled via SecOCDevErrorDetect, the SecOC module shall report the DET error SECOC_E_CRYPTO_FAILURE when it is finally not able to get the required security services for authentication/verification from the CSM.
⌋ (SRS_BSW_00337, SRS_BSW_00350, SRS_BSW_00385)

[SWS_SecOC_00167]⌈
If DET reporting is enabled via SecOCDevErrorDetect, the SecOC module shall report the DET error SECOC_E_FRESHNESS_VALUE_AT_LIMIT when a

Freshness Value has reached its limit (see SWS_SecOC_00017 and SWS_SecOC_00168).
] (SRS_BSW_00337, SRS_BSW_00350, SRS_BSW_00385)

8 API specification

8.1 Imported types

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

[SWS_SecOC_00103] Imported Types
⌈

Module	*Imported Type*
Cal	Cal_AsymPrivateKeyType
	Cal_AsymPublicKeyType
	Cal_ConfigIdType
	Cal_MacGenerateCtxBufType
	Cal_MacVerifyCtxBufType
	Cal_ReturnType
	Cal_SignatureGenerateCtxBufType
	Cal_SignatureVerifyCtxBufType
	Cal_SymKeyType
	Cal_VerifyResultType
ComStack_Types	BufReq_ReturnType
	PduIdType
	PduInfoType
	PduLengthType
	RetryInfoType
Csm	Csm_AsymPrivateKeyType
	Csm_AsymPublicKeyType
	Csm_ConfigIdType
	Csm_SymKeyType
	Csm_VerifyResultType
NvM	NvM_BlockIdType
	NvM_RequestResultType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋ (SRS_BSW_00301)

8.2 Type definitions

8.2.1 SecOC_ConfigType

[SWS_SecOC_00104] SecOC_ConfigType
⌈

Name:	`SecOC_ConfigType`	
Type:	`Structure`	
Range:	`implementation specific`	The content of the configuration data structure is implementation specific.
Description:	Configuration data structure of SecOC module	

⌋ (SRS_SecOC_00001, SRS_SecOC_00003)

8.2.2 SecOC_StateType

[SWS_SecOC_00162] SecOC_StateType
⌈

Name:	SecOC_StateType	
Type:	Enumeration	
Range:	SECOC_UNINIT	SecOC module is not initialized
	SECOC_INIT	SecOC module is initialized
Description:	States of the SecOC module	

⌋ (SRS_SecOC_00005)

8.2.3 SecOC_AlignType

[SWS_SecOC_00154] SecOC_AlignType
⌈

Name:	SecOC_AlignType
Type:	<maxAlignScalarType>
Description:	A scalar type which has maximum alignment restrictions on the given platform. This value is configured by "SecOCMaxAlignScalarType". <maxAlignScalarType> can be e.g. uint8, uint16 or uint32. This type shall be consistent with Csm_AlignType (if CSM is used) or Cal_AlignType (if CAL is used).

⌋ (SRS_SecOC_00005)

8.2.4 SecOC_KeyType

[SWS_SecOC_00105] SecOC_KeyType
⌈

Name:	SecOC_KeyType		
Type:	Structure		
Element:	uint32	length	This element contains the length of the key stored in element 'data'.
	SecOC_AlignType[SECOC_KEY_MAX_SIZE]	data	This element contains the key data or a key handle.
Description:	Data structure to refer to key data or a key handle.		

⌋ (SRS_SecOC_00005)

8.2.5 SecOC_VerificationResultType

[SWS_SecOC_00149] SecOC_VerificationResultType
⌈

Name:	SecOC_VerificationResultType	
Type:	Enumeration	
Range:	SECOC_VERIFICATIONSUCCESS	Verification successful
	SECOC_VERIFICATIONFAILURE	Verification not successful
	SECOC_FRESHNESSFAILURE	Verification not successful because of wrong freshness value.

Description:	Enumeration to indicate verification results.

] (SRS_SecOC_00022)

Note: SECOC_FRESHNESSFAILURE is only applicable if the complete freshness value has been transmitted.

8.2.6 SecOC_VerificationStatusType

[SWS_SecOC_00160] SecOC_VerificationStatusType
[

Name:	`SecOC_VerificationStatusType`		
Type:	`Structure`		
Element:	`uint16`	`freshnessValueID`	Identifier of the Freshness Value which resulted in the Verification Status
	`SecOC_VerificationResultType`	`verificationStatus`	Result of verification attempt: SECOC_VERIFICATIONSUCCESS = Verification successful SECOC_VERIFICATIONFAILURE = Verification not successful SECOC_FRESHNESSFAILURE = Verification not successful because of wrong freshness value
Description:	Data structure to bundle the status of a verification attempt for a specific Freshness Value.		

] (SRS_SecOC_00022)

8.3 Function definitions

8.3.1 SecOC_Init

Add the following function:

[SWS_SecOC_00106] SecOC_Init
[

Service name:	SecOC_Init	
Syntax:	`void SecOC_Init( const SecOC_ConfigType* config )`	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	config	Pointer to a selected configuration structure
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the the SecOC module. Successful initialization leads to state SecOC_INIT.	

] (SRS_BSW_00101, SRS_BSW_00323, SRS_BSW_00358, SRS_BSW_00359, SRS_BSW_00414, , SRS_SecOC_00006)

[SWS_SecOC_00161] SecOC_DeInit
⌈

Service name:	SecOC_DeInit

Syntax:	`void SecOC_DeInit(`
	`  void`
	`)`

Service ID[hex]:	0x05
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service stops the secure onboard communication. All buffered I-PDU are removed and have to be obtained again, if needed, after SecOC_Init has been called. By a call to SecOC_DeInit the AUTOSAR SecOC module is put into a not initialized state (SecOC_UNINIT).

⌋ (SRS_BSW_00323, SRS_BSW_00359, SRS_SecOC_00006, SRS_SecOC_00020)

[SWS_SecOC_00157]⌈
Within SecOC_DeInit the module shall clear all internal global variables and the buffers of the SecOC I-PDUs.
⌋ (SRS_BSW_00323, SRS_SecOC_00006)

[SWS_SecOC_00156]⌈
Within SecOC_DeInit the module shall store all Freshness Values and all key related information (e.g. key handles) to NVRAM, so that all values could be restored to the status they had before SecOC_DeInit has been called.
⌋ (SRS_BSW_00323, SRS_SecOC_00006, SRS_SecOC_00020)

8.3.2 SecOC_GetVersionInfo

[SWS_SecOC_00107] SecOC_GetVersionInfo
⌈

Service name:	SecOC_GetVersionInfo	

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

Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Returns the version information of this module.	

] (SRS_BSW_00323, SRS_BSW_00359, SRS_BSW_00407, SRS_BSW_00369, SRS_BSW_00003, SRS_BSW_00402)

8.3.3 SecOC_Transmit

[SWS_SecOC_00112] SecOC_Transmit
[

Service name:	SecOC_Transmit	
Syntax:	`Std_ReturnType SecOC_Transmit( PduIdType id, const PduInfoType* info )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant for the same PDU-ID. Reentrant for different PDU-ID.	
Parameters (in):	id	ID of the Authentic I-PDU to be transmitted
	info	A pointer to a structure with Authentic I-PDU related data that shall be transmitted: data length and pointer to I-SDU buffer
Parameters (inout):	None	
Parameters (out):	None	
Return value:	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.
Description:	Service is called by the PduR to request authentication and transmission of an Authentic I-PDU.	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_BSW_00369, SRS_BSW_00449)
For detailed description, see Section 7.4.

8.3.4 SecOC_CancelTransmit

[SWS_SecOC_00113] SecOC_CancelTransmit
[

Service name:	SecOC_CancelTransmit	
Syntax:	`Std_ReturnType SecOC_CancelTransmit( PduIdType id )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant for the same PDU-ID. Reentrant for different PDU-ID.	
Parameters (in):	id	ID of the Authentic I-PDU to be cancelled
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation request was executed successfully by the SecOC module. E_NOT_OK: Cancellation request was rejected.
Description:	Service is called by the PduR to request the cancellation of an authentication and transmission of an Authentic I-PDU.	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_BSW_00449, SRS_SecOC_00012)

8.3.5 SecOC_AssociateKey

[SWS_SecOC_00116] SecOC_AssociateKey
[

Service name:	SecOC_AssociateKey	
Syntax:	`Std_ReturnType SecOC_AssociateKey( uint8 keyID, const SecOC_KeyType* keyPtr )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	keyID	Identifier of a local key slot
	keyPtr	This element points to the key data or a key handle
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request successful E_NOT_OK: request failed
Description:	Service associates a given key value to a given key id (see also parameter SecOCKeyID).	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_BSW_00449, SRS_SecOC_00003)

8.3.6 SecOC_FreshnessValueRead

[SWS_SecOC_00117] SecOC_FreshnessValueRead
[

Service name:	SecOC_FreshnessValueRead	
Syntax:	`Std_ReturnType SecOC_FreshnessValueRead( uint16 freshnessValueID, uint64* counterValue )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant for the same FreshnessValueID. Reentrant for different FreshnessValueIDs	
Parameters (in):	freshnessValueID	Identifier of a specific Freshness Value
Parameters (inout):	None	
Parameters (out):	counterValue	Holds the current value of the counter
Return value:	Std_ReturnType	E_OK: request successful E_NOT_OK: request failed
Description:	This service is used to read a specific Freshness Value value residing in the SecOC module.	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_BSW_00449, SRS_SecOC_00002)

8.3.7 SecOC_FreshnessValueWrite

[SWS_SecOC_00118] SecOC_FreshnessValueWrite[

Service name:	SecOC_FreshnessValueWrite	
Syntax:	`Std_ReturnType SecOC_FreshnessValueWrite( uint16 freshnessValueID, uint64 counterValue )`	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant for the same FreshnessValueID. Reentrant for different FreshnessValueIDs	
Parameters (in):	freshnessValueID	Identifier of a specific Freshness Value
	counterValue	Holds the counter value to be written
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request successful E_NOT_OK: request failed
Description:	This service is used to write a specific Freshness Value residing in the SecOC module.	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_BSW_00449, SRS_SecOC_00002)

8.3.8 Optional Interfaces

This chapter defines all external interfaces that are required to fulfil an optional functionality of the module.

[SWS_SecOC_00122] SecOC_VerifyStatusOverride[

Service name:	SecOC_VerifyStatusOverride	
Syntax:	`Std_ReturnType SecOC_VerifyStatusOverride( uint16 freshnessValueID, uint8 overrideStatus, uint8 numberOfMessagesToOverride )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant for the same FreshnessValueID. Reentrant for different FreshnessValueIDs	
Parameters (in):	freshnessValueID	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	0 = Override VerifyStatus to "Fail" until further notice 1 = Override VerifyStatus to "Fail" until NumberOfMessagesToOverride is reached 2 = Cancel Override of VerifyStatus
	numberOfMessagesToOverride	Number of sequential VerifyStatus to override when using a specific counter for authentication verification. This is only considered when OverrideStatus is equal to 1
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request successful E_NOT_OK: request failed
Description:	This service provides the ability to override the VerifyStatus with â€œFailâ€•	

	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. Note: When overriding the VerifyStatus, the CSM shall still be used to validate authentication of the data making up an I-PDU. This service is optional.

] (SRS_BSW_00323, SRS_BSW_00357, SRS_BSW_00449, SRS_SecOC_00017)

8.4 Call-back notifications

8.4.1 SecOC_RxIndication

[SWS_SecOC_00124] SecOc_RxIndication
⌈

Service name:	SecOC_RxIndication	

Syntax:	`void SecOC_RxIndication(`
	`  PduIdType RxPduId,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x42	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	RxPduId	ID of the received I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication interface module.	

] (SRS_BSW_00323, SRS_BSW_00359, SRS_SecOC_00012)

8.4.2 SecOC_TpRxIndication

[SWS_SecOC_00125] SecOc_TpRxIndication
⌈

Service name:	SecOC_TpRxIndication	

Syntax:	`void SecOC_TpRxIndication(`
	`  PduIdType id,`
	`  Std_ReturnType result`
	`)`	

Service ID[hex]:	0x45	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the received I-PDU.
	result	Result of the reception.
Parameters (inout):	None	
Parameters (out):	None	

Return value:	None
Description:	Called after an I-PDU has been received via the TP API, the result indicates whether the transmission was successful or not.

] (SRS_BSW_00323, SRS_BSW_00359, SRS_BSW_00449, SRS_SecOC_00012)

8.4.3 SecOC_TxConfirmation

[SWS_SecOC_00126] SecOc_TxConfirmation
⌈

Service name:	SecOC_TxConfirmation	

Syntax:	`void SecOC_TxConfirmation(`
	`  PduIdType TxPduId`
	`)`	

Service ID[hex]:	0x40	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the I-PDU that has been transmitted.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The lower layer communication interface module confirms the transmission of an I-PDU.	

] (SRS_BSW_00323, SRS_BSW_00359, SRS_SecOC_00012)

8.4.4 SecOC_TpTxConfirmation

[SWS_SecOC_00152] SecOc_TpTxConfirmation ⌈
⌈

Service name:	SecOC_TpTxConfirmation	

Syntax:	`void SecOC_TpTxConfirmation(`
	`  PduIdType id,`
	`  Std_ReturnType result`
	`)`	

Service ID[hex]:	0x48	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the transmitted I-PDU.
	result	Result of the transmission of the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This function is called after the I-PDU has been transmitted on its network, the result indicates whether the transmission was successful or not.	

] (SRS_BSW_00323, SRS_BSW_00359, SRS_BSW_00449, SRS_SecOC_00012)

8.4.5 SecOC_TriggerTransmit

[SWS_SecOC_00127] SecOc_TriggerTransmit⌈

[

Service name:	SecOC_TriggerTransmit	
Syntax:	`Std_ReturnType SecOC_TriggerTransmit( PduIdType TxPduId, PduInfoType* PduInfoPtr )`	
Service ID[hex]:	0x41	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the SDU that is requested to be transmitted.
Parameters (inout):	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLengh. On return, the service will indicate the length of the copied SDU data in SduLength.
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: SDU has been copied and SduLength indicates the number of copied bytes. E_NOT_OK: No SDU data has been copied. PduInfoPtr must not be used since it may contain a NULL pointer or point to invalid data.
Description:	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_BSW_00449, SRS_SecOC_00012)

8.4.6 SecOC_CopyRxData

[SWS_SecOC_00128] SecOc_CopyRxData

[

Service name:	SecOC_CopyRxData	
Syntax:	`BufReq_ReturnType SecOC_CopyRxData( PduIdType id, const PduInfoType* info, PduLengthType* bufferSizePtr )`	
Service ID[hex]:	0x44	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the received I-PDU.
	info	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.
Parameters (inout):	None	
Parameters (out):	bufferSizePtr	Available receive buffer after data has been copied.
Return value:	BufReq_ReturnType	BUFREQ_OK: Data copied successfully BUFREQ_E_NOT_OK: Data was not copied because an error occurred.
Description:	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.

] (SRS_BSW_00323, SRS_BSW_00357, SRS_SecOC_00012)

8.4.7 SecOC_CopyTxData

[SWS_SecOC_00129] SecOc_CopyTxData[
[

Service name:	SecOC_CopyTxData	

Syntax:	`BufReq_ReturnType SecOC_CopyTxData(`
	`  PduIdType id,`
	`  const PduInfoType* info,`
	`  RetryInfoType* retry,`
	`  PduLengthType* availableDataPtr`
	`)`	

Service ID[hex]:	0x43	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the transmitted I-PDU.
	info	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	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 indicates that this API call shall copy previously copied data in order to recover from an error. In this case TxTpDataCnt specifies the offset in bytes from the current data copy position.
Parameters (inout):	None	
Parameters (out):	availableDataPtr	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 value:	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.
Description:	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.	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_SecOC_00012)

8.4.8 SecOC_StartOfReception

[SWS_SecOC_00130] SecOc_StartOfReception[
[

Service name:	SecOC_StartOfReception	

Syntax:	`BufReq_ReturnType SecOC_StartOfReception(`
	`  PduIdType id,`
	`  const PduInfoType* info,`
	`  PduLengthType TpSduLength,`
	`  PduLengthType* bufferSizePtr`
	`)`	

Service ID[hex]:	0x46	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the I-PDU.
	info	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	Total length of the N-SDU to be received.
Parameters (inout):	None	
Parameters (out):	bufferSizePtr	Available receive buffer in the receiving module. This parameter will be used to compute the Block Size (BS) in the transport protocol module.
Return value:	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.
Description:	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).	

] (SRS_BSW_00323, SRS_BSW_00357, SRS_SecOC_00012)

8.4.9 CSM callback interfaces

[SWS_SecOC_00012] ⌈
If the SecOC module uses the Csm module asynchronously to calculate or verify the authenticator, SecOC shall provide callback functions according to Csm_CallbackType.
⌋ (SRS_BSW_00457, SRS_SecOC_00003)

8.5 Scheduled functions

8.5.1 SecOC_MainFunction

[SWS_SecOC_00131] SecOC_MainFunction
⌈

Service name:	SecOC_MainFunction

Syntax:	`void SecOC_MainFunction(`
	`  void`
	`)`

Service ID[hex]:	0x06
Description:	This function performs the processing of the SecOC module's authentication and verification processing.

⌋ (SRS_BSW_00373, SRS_BSW_00425)

[SWS_SecOC_00132] ⌈
If the SecOC module was not previously initialized with a call to SecOC_Init, then a call to SecOC_MainFunction shall simply return.
⌋ (SRS_SecOC_00005)

[SWS_SecOC_00133] ⌈
The cycle time of the `SecOC_MainFunction` is configured by the parameter SecOCMainFunctionPeriod.
⌋ (SRS_SecOC_00025)

[SWS_SecOC_00134] ⌈
If SecOC_MainFunction is scheduled, the SecOC shall firstly check if there are new Authentic I-PDUs to be authenticated or new Secured I-PDUs to be verified. If yes the SecOC module shall process the authentication or verification of each of the IPDUs identified as new subsequently in the very same main function call.

⌋ (SRS_SecOC_00025)

[SWS_SecOC_00135] ⌈
For each newly authenticated Authentic I-PDU, the SecOC module shall immediately trigger the transmission of the Secured I-PDU at the lower layer module by calling the PduR.

⌋ (SRS_SecOC_00025)

[SWS_SecOC_00136] ⌈

For each newly successfully verified Secured I-PDU, the SecOC module shall immediately pass the Authentic I-PDU to the upper layer communication module by calling PduR_SecOCRxIndication for the Authentic I-PDU.

] (SRS_SecOC_00025)

8.6 Expected Interfaces

8.6.1 Mandatory Interfaces

This chapter defines all external interfaces that are required to fulfill the core functionality of the module.

[SWS_SecOC_00137] Mandatory Interfaces
[

API function	*Description*
NvM_GetErrorStatus	Service to read the block dependent error/status information.
NvM_ReadBlock	Service to copy the data of the NV block to its corresponding RAM block.
NvM_WriteBlock	Service to copy the data of the RAM block to its corresponding NV block.
PduR_SecOCCancelTransmit	Requests cancellation of an ongoing transmission of an I-PDU in a lower layer communication interface or transport protocol module.
PduR_SecOCRxIndication	Indication of a received I-PDU from a lower layer communication interface module.
PduR_SecOCTransmit	Requests transmission of an I-PDU.
PduR_SecOCTxConfirmation	The lower layer communication interface module confirms the transmission of an I-PDU.

] (SRS_BSW_00384)

8.6.2 Optional Interfaces

[SWS_SecOC_00138] Optional Interfaces
[

API function	*Description*
Cal_MacGenerateFinish	This function shall be used to finish the MAC generation service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The MAC computation is done by the underlying primitive.
Cal_MacGenerateStart	This function shall be used to initialize the MAC generate service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified

	by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.
Cal_MacGenerateUpdate	This function shall be used to feed the MAC generate service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The MAC computation is done by the underlying primitive.
Cal_MacVerifyFinish	This function shall be used to finish the MAC verification service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The MAC computation is done by the underlying primitive.The MAC computation is done by the underlying primitive.
Cal_MacVerifyStart	This function shall be used to initialize the MAC verify service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.
Cal_MacVerifyUpdate	This function shall be used to feed the MAC verification service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The MAC computation is done by the underlying primitive.The MAC computation is done by the underlying primitive.
Cal_SignatureGenerateFinish	This function shall be used to finish the signature generation service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The signature computation is done by the underlying primitive.
Cal_SignatureGenerateStart	This function shall be used to initialize the signature generate service of the CAL module.

	The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.
Cal_SignatureGenerateUpdate	This function shall be used to feed the signature generation service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The signature computation is done by the underlying primitive.
Cal_SignatureVerifyFinish	This function shall be used to finish the signature verification service. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Finish of the primitive which is identified by the "cfgId", and return the value returned by that function. If Cpl_<Primitive>Finish returned successfully, the function shall set the state of this service to "idle", and store this state in the context buffer. The signature computation is done by the underlying primitive.
Cal_SignatureVerifyStart	This function shall be used to initialize the signature verify service of the CAL module. The function shall initialize the context buffer given by "contextBuffer", call the function Cpl_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cpl_<Primitive>Start returned successfully, the function shall set the state of this service to "active", and store this state in the context buffer.
Cal_SignatureVerifyUpdate	This function shall be used to feed the signature verification service with the input data. If the service state given by the context buffer is "idle", the function has to return with "CAL_E_NOT_OK". Otherwise, this function shall call the function Cpl_<Primitive>Update of the primitive which is identified by the "cfgId", and return the value returned by that function. The signature computation is done by the underlying primitive.
Csm_MacGenerateFinish	This interface shall be used to finish the MAC generation service. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.
Csm_MacGenerateStart	This interface shall be used to initialize the MAC generate service of the CSM module.

	If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".
Csm_MacGenerateUpdate	This interface shall be used to feed the MAC generate service with the input data. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.
Csm_MacVerifyFinish	This interface shall be used to finish the MAC verification service. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.
Csm_MacVerifyStart	This interface shall be used to initialize the MAC verify service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".
Csm_MacVerifyUpdate	This interface shall be used to feed the MAC verification service with the input data. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The MAC computation is done by the underlying primitive.
Csm_SignatureGenerateFinish	This interface shall be used to finish the signature generation service. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.
Csm_SignatureGenerateStart	This interface shall be used to initialize the signature generate service of the CSM module.

	If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".
Csm_SignatureGenerateUpdate	This interface shall be used to feed the signature generation service with the input data. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.
Csm_SignatureVerifyFinish	This interface shall be used to finish the signature verification service. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Finish of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.
Csm_SignatureVerifyStart	This interface shall be used to initialize the signature verify service of the CSM module. If the service state is "active", the function shall return with "CSM_E_BUSY". Otherwise, this function shall store the given configuration information which is identified by "cfgId", call the function Cry_<Primitive>Start of the primitive which is identified by the "cfgId" and return the value returned by that function. If Cry_<Primitive>Start returned successfully, the service state has to be set to "active".
Csm_SignatureVerifyUpdate	This interface shall be used to feed the signature verification service with the input data. If the service state is "idle", the function has to return with "E_NOT_OK". Otherwise, this function shall call the function Cry_<Primitive>Update of the primitive which is identified by the stored configuration information and return the value returned by that function. The signature computation is done by the underlying primitive.
Det_ReportError	Service to report development errors.

] (SRS_BSW_00384)

8.6.3 Configurable Interfaces

8.6.3.1 SecOC_VerificationStatusCallout

[SWS_SecOC_00119] [If configured by SecOCVerificationStatusCallout (see ECUC_SecOC_00004), the SecOC module shall invoke a callout function to notify other modules on the verification status of the most recently received Secured I-PDU.

Service name:	SecOC_VerificationStatusCallout	
Syntax:	`void SecOC_VerificationStatusCallout( SecOC_VerificationStatusType verificationStatus )`	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant for the same FreshnessValueID. Reentrant for different FreshnessValueIDs	
Parameters (in):	verificationStatus	Data structure to bundle the status of a verification attempt for a specific Freshness Value.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
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 are propagated to modules - None: No Verification Status is propagated	

] (SRS_BSW_00359, SRS_SecOC_00017)

Note: The argument freshnessValueID allows for unambiguously identifying the Secured I-PDU that was subject of the verification attempt. Since each Secured I-PDU has at least one but possibly two related Freshness Value IDs (i.e. a Secured I-PDU may have a Secondary Freshness Value ID), SecOC_VerificationStatusCallout is able to indicate for which of the freshness values the verification attempt has been carried out.

Note: Any module that is configured to be notified by the means of SecOC_VerificationStatusCallout has to implement a target function that is conforming to the above signature. The name of the target function listed above are not fixed. The name could be configured by means of the parameter SecOCVerificationStatusCallout.

8.7 Service Interfaces

This chapter defines the AUTOSAR Interfaces of the SecOC Service (<MA>).

The definitions in this section are interpreted to be in `ARPackage AUTOSAR/Services/<MA>`.

8.7.1 Overview

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

8.7.2 Sender Receiver Interfaces

8.7.2.1 Verification Status Service

[SWS_SecOC_00141]⌈

Name	VerificationStatus	
Comment	This service realizes a notification service that is used to propagate the status of each authentication attempt from the SecOC module to the application layer. This service can be configured such that: - Only "False" Verification Status is propagated to the application layer - Both "True" and "False" Verification Status are propagated to the application layer - No Verification Status is propagated to the application layer	
IsService	true	
Variation	--	
Data Elements	verificationStatus	
	Type	SecOC_VerificationStatusType
	Variation	--

⌋ (SRS_SecOC_00022)

Note: The SecOC_VerificationStatusService is used to propagate the status of each verification attempt from the SecOC module to an arbitrary number of application software components. It can be used to continuously monitor the number of failed verification attempts for a given FreshnessValueID or a set thereof and would allow setting up a security management system/intrusion detection system that is able to detect an attack flood and react with adequate dynamic countermeasures.

[SWS_SecOC_00148]⌈
SecOC shall define a provide port for the SecOC_VerificationStatusService interface and call the generated Rte function as configured by the parameter SecOCVerificationStatusPropagationMode. The sender/receiver interface shall be defined as standard interface.

⌋(SRS_SecOC_00022)

8.7.3 Client Server Interfaces

8.7.3.1 Key Management Service

[SWS_SecOC_00139]⌈

Name	KeyManagement
Comment	Key Management Service of SecOC
IsService	true
Variation	--

Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

AssociateKey		
Comments	Associates a given key value to a given key id (see also parameter SecOCKeyID).	
Variation	--	
Parameters	keyId	
	Comment	Identifier of a local key slot
	Type	uint8
	Variation	--
	Direction	IN
	keyPtr	
	Comment	Comment This element points to the key data or a key handle
	Type	SecOC_KeyType
	Variation	--
	Direction	IN
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--

] (SRS_SecOC_00003)

8.7.3.2 Counter Management Service

[SWS_SecOC_00140][

Name	CounterManagement	
Comment	Counter Management Service of SecOC	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

FreshnessValueRead

Comments	This service is used to read a specific Freshness Value value residing in the SecOC module.	
Variation	--	
Parameters	freshnessValueId	
	Comment	Identifier of a specific Freshness Value
	Type	uint16
	Variation	--
	Direction	IN
	counterValue	
	Comment	Holds the current value of the counter
	Type	uint64
	Variation	--
	Direction	OUT
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--
FreshnessValueWrite		
Comments	This service is used to write a specific Freshness Value residing in the SecOC module.	
Variation	--	
Parameters	freshnessValueId	
	Comment	Identifier of a specific Freshness Value
	Type	uint16
	Variation	--
	Direction	IN
	counterValue	
	Comment	Holds the counter value to be written
	Type	uint64
	Variation	--
	Direction	IN
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--

] (SRS_SecOC_00002)

8.7.3.3 Verification Status Configuration Service

[SWS_SecOC_00142][

Name	VerifyStatusConfiguration	
Comment	Verify Status Configuration Service of SecOC	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

VerifyStatusOverride		
Comments	This service provides the ability to override the VerifyStatus with "Fail" 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. Note: When overriding the VerifyStatus, the CSM shall still be used to validate authentication of the data making up an I-PDU. This service is optional.	
Variation	--	
Parameters	freshnessValueId	
	Comment	Identifier of the Freshness Value which resulted in the AuthenticationStatus
	Type	uint16
	Variation	--
	Direction	IN
	overrideStatus	
	Comment	0 = Override VerifyStatus to "Fail" until further notice 1 = Override VerifyStatus to "Fail" until NumberOfMessagesToOverride is reached 2 = Cancel Override of VerifyStatus
	Type	uint8
	Variation	--
	Direction	IN

	numberOfMessagesToOverride	
	Comment	Number of sequential VerifyStatus to override when using a specific counter for authentication verification. This is only considered when OverrideStatus is equal to 1
	Type	uint8
	Variation	--
	Direction	IN
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--

] (SRS_SecOC_00017)

9 Sequence diagrams

The sequence diagrams in the following sections show interactions between the SecOC module, the PDuR and the upper layer and lower layer communication modules. These sequences serve as examples to express the different kinds of interactions that are served by the SecOC module for authentication and verification.

Note: The examples show the interaction with distinct bus interface (e.g FrIf), transport protocol module (e.g. CanTp) or upper layer communication module (e.g. COM) only. However, they are valid for other bus interfaces, transport protocol modules and upper layer communication modules as well.

Note: The examples use the following color scheme to distinguish the handling of Authentic I-PDUs and Secured I-PDUs: Operation that refer to Authentic I-PDUs are denoted in **blue** and operations that refer to Secured I-PDUs are denoted in **green**.

9.1 Authentication of outgoing PDUs

9.1.1 Authentication during direct transmission

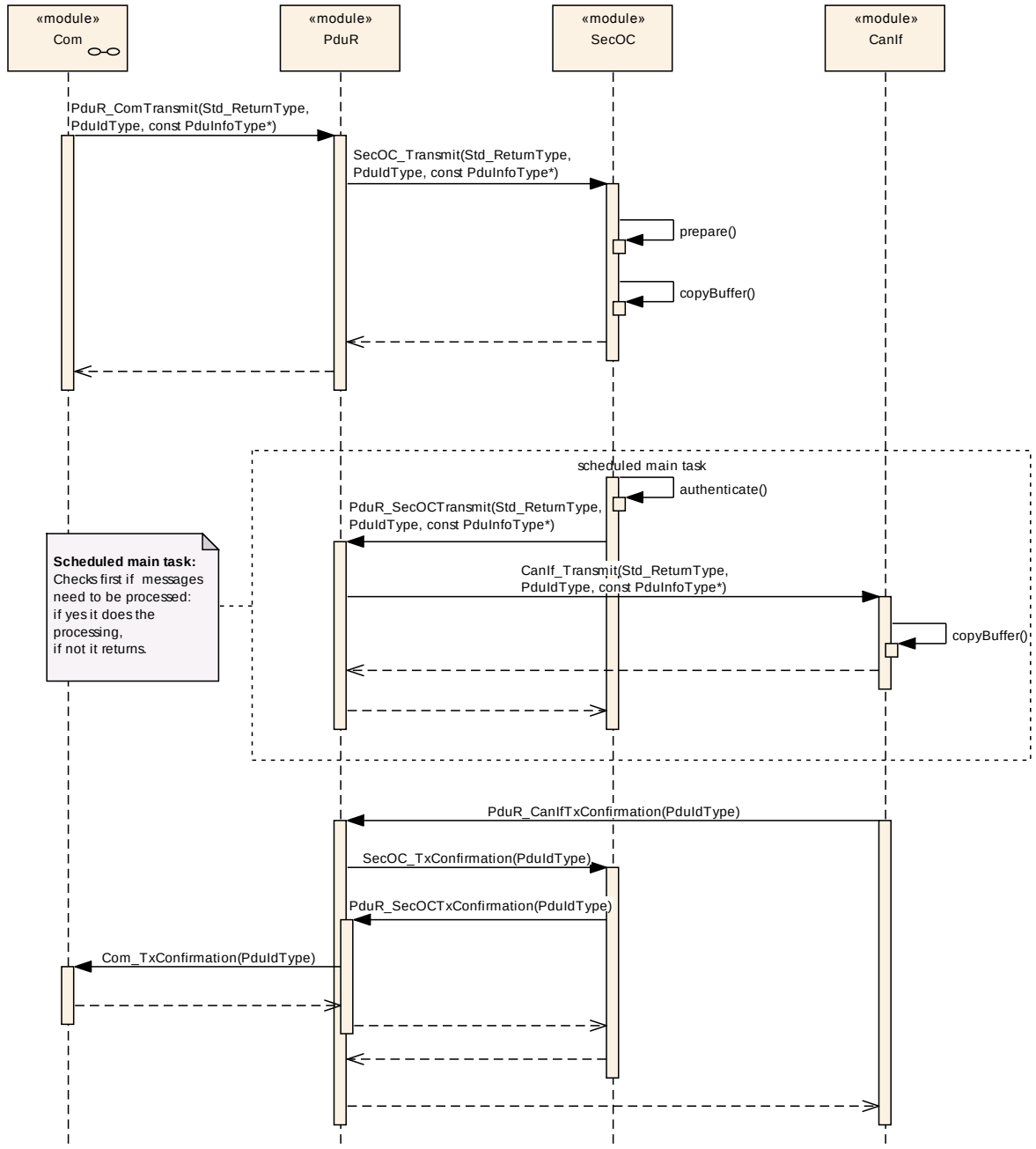

Figure 9 Authentication during direct transmission

9.1.2 Authentication during triggered transmission

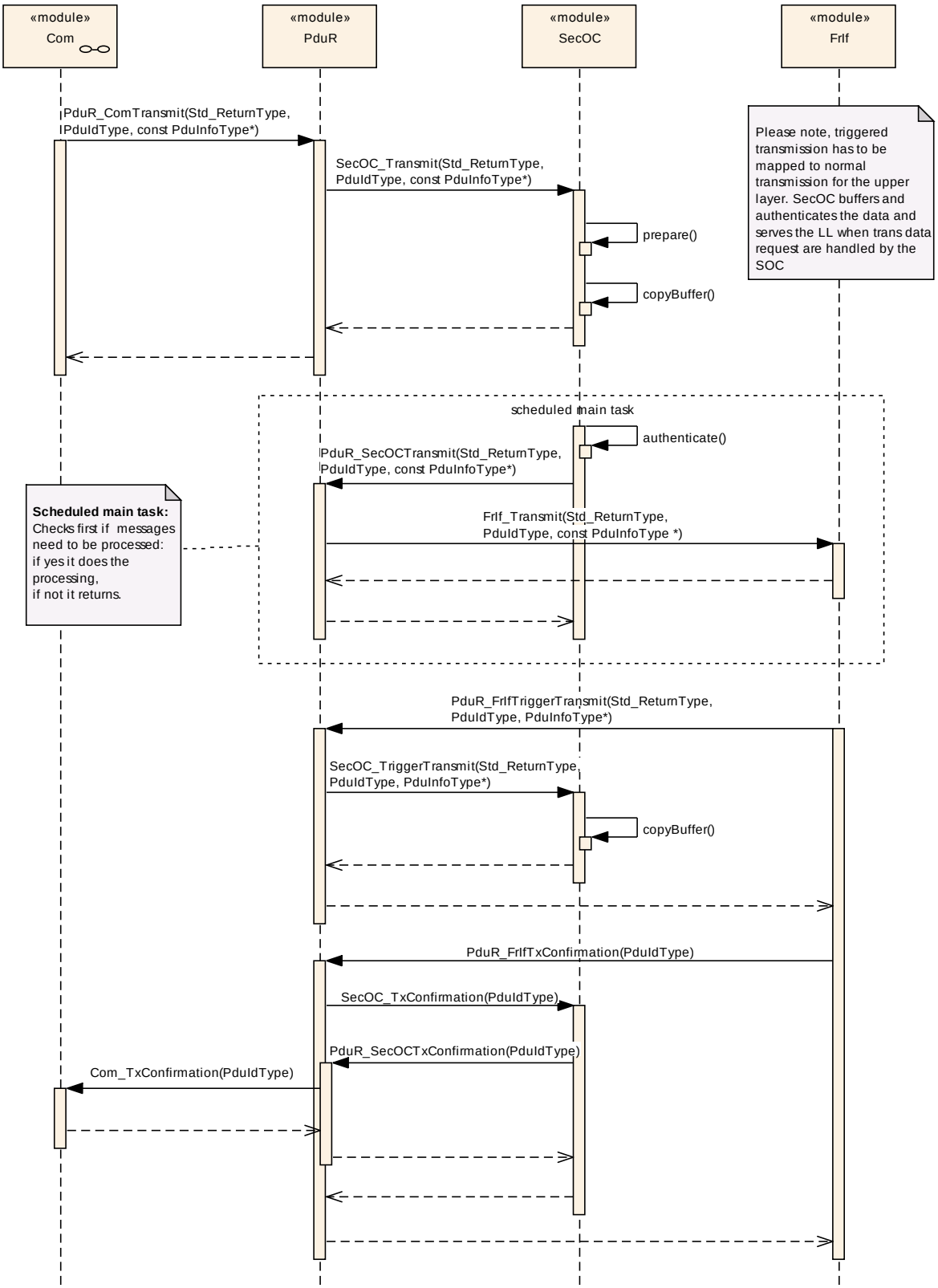

Figure 10 Authentication during Triggered Transmission

9.1.3 Authentication during transport protocol transmission

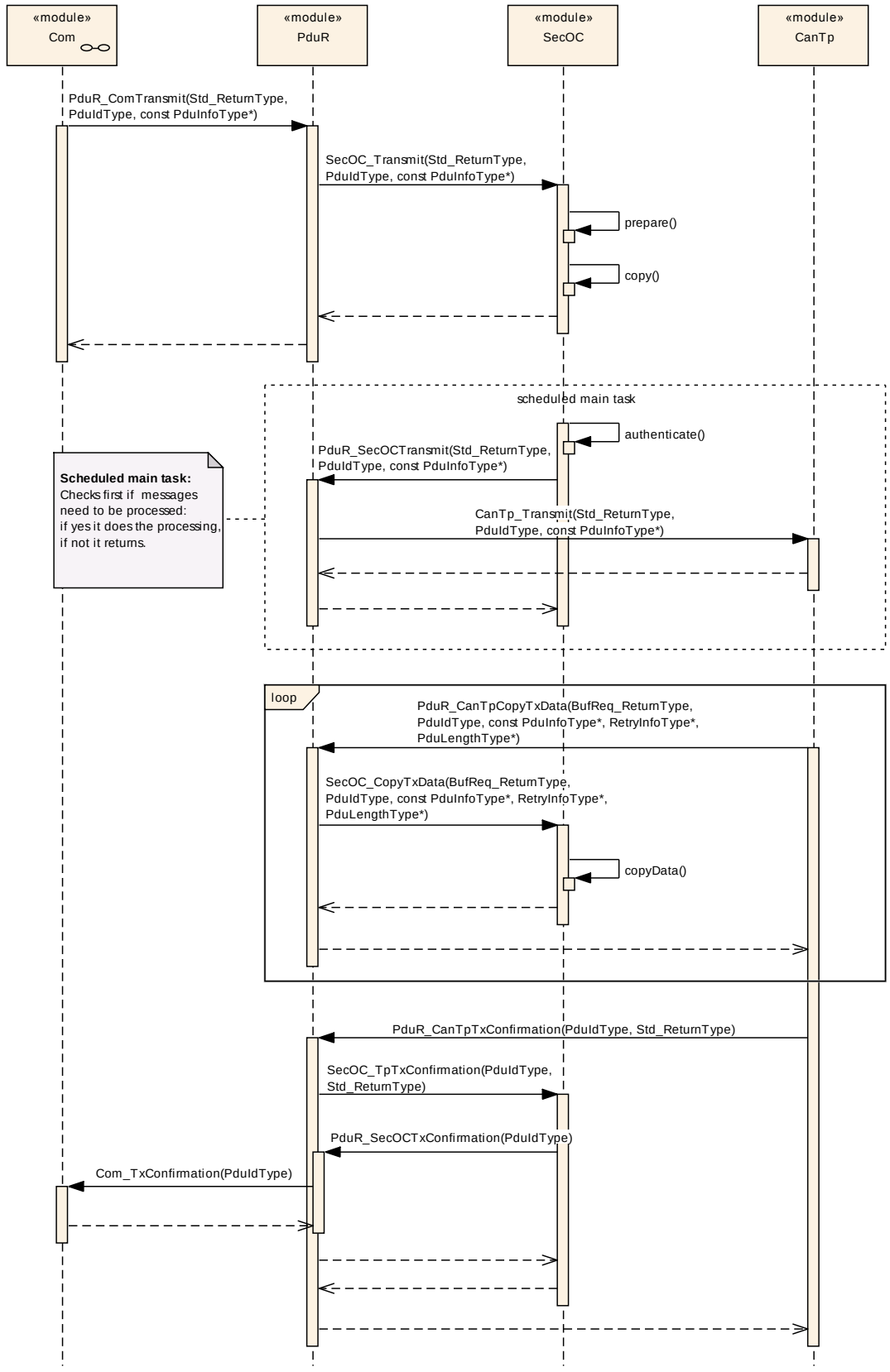

Figure 11 Authentication during TP transmission

9.2 Verification of incoming PDUs

9.2.1 Verification during direct reception

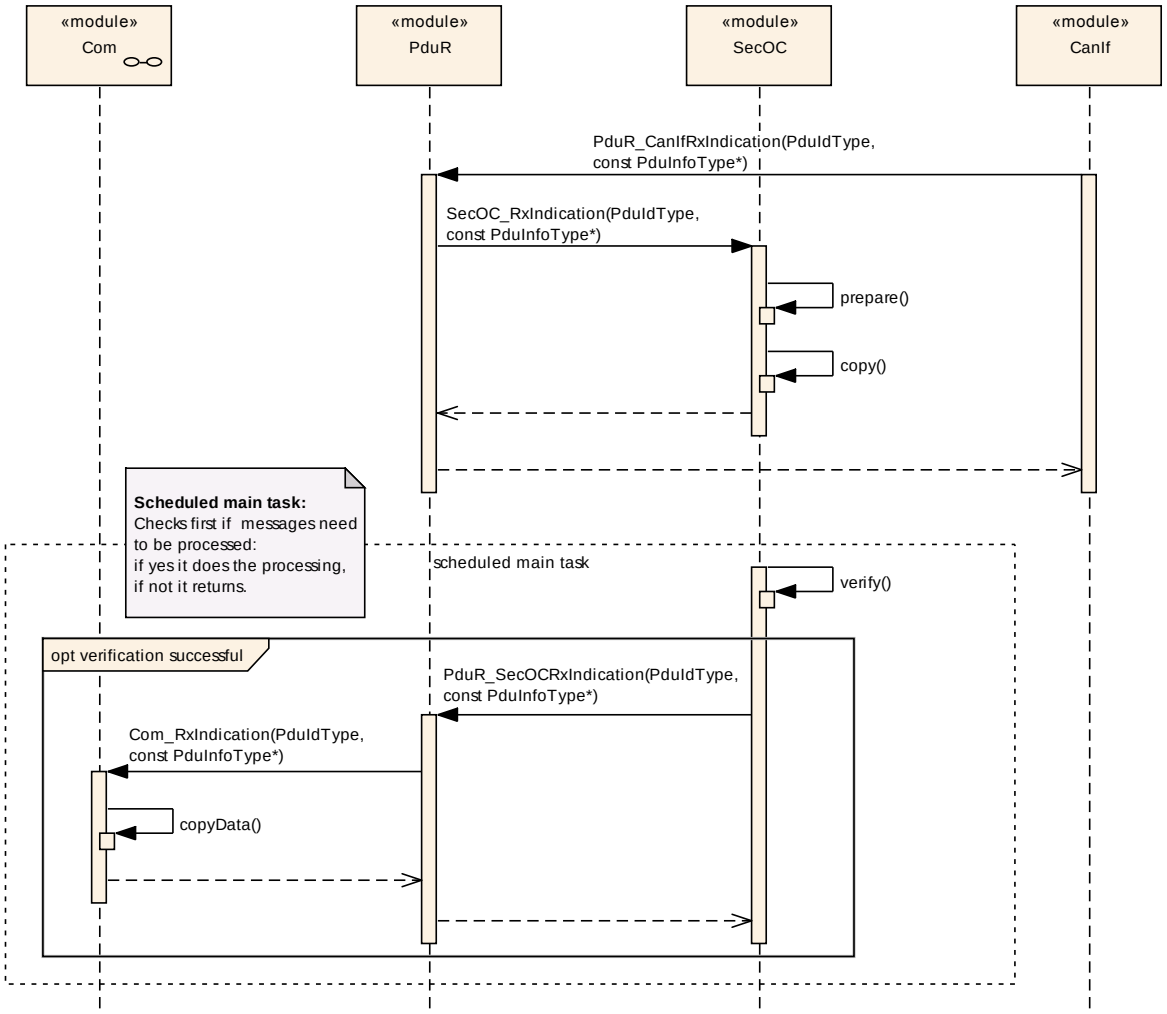

Figure 12 Verification during direct reception

9.2.2 Verification during transport protocol reception

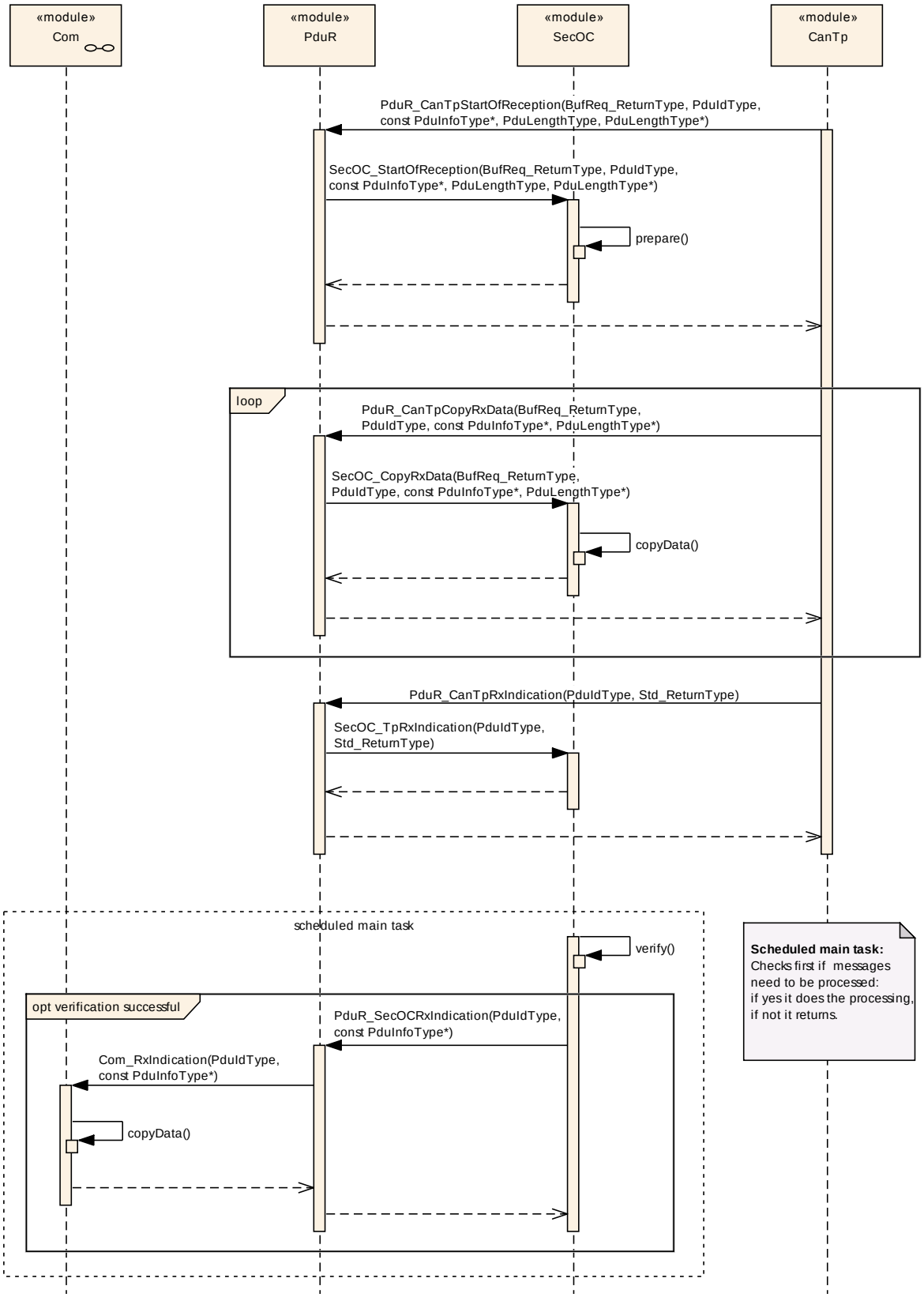

Figure 13 Verification during transport protocol reception

9.3 Re-authentication Gateway

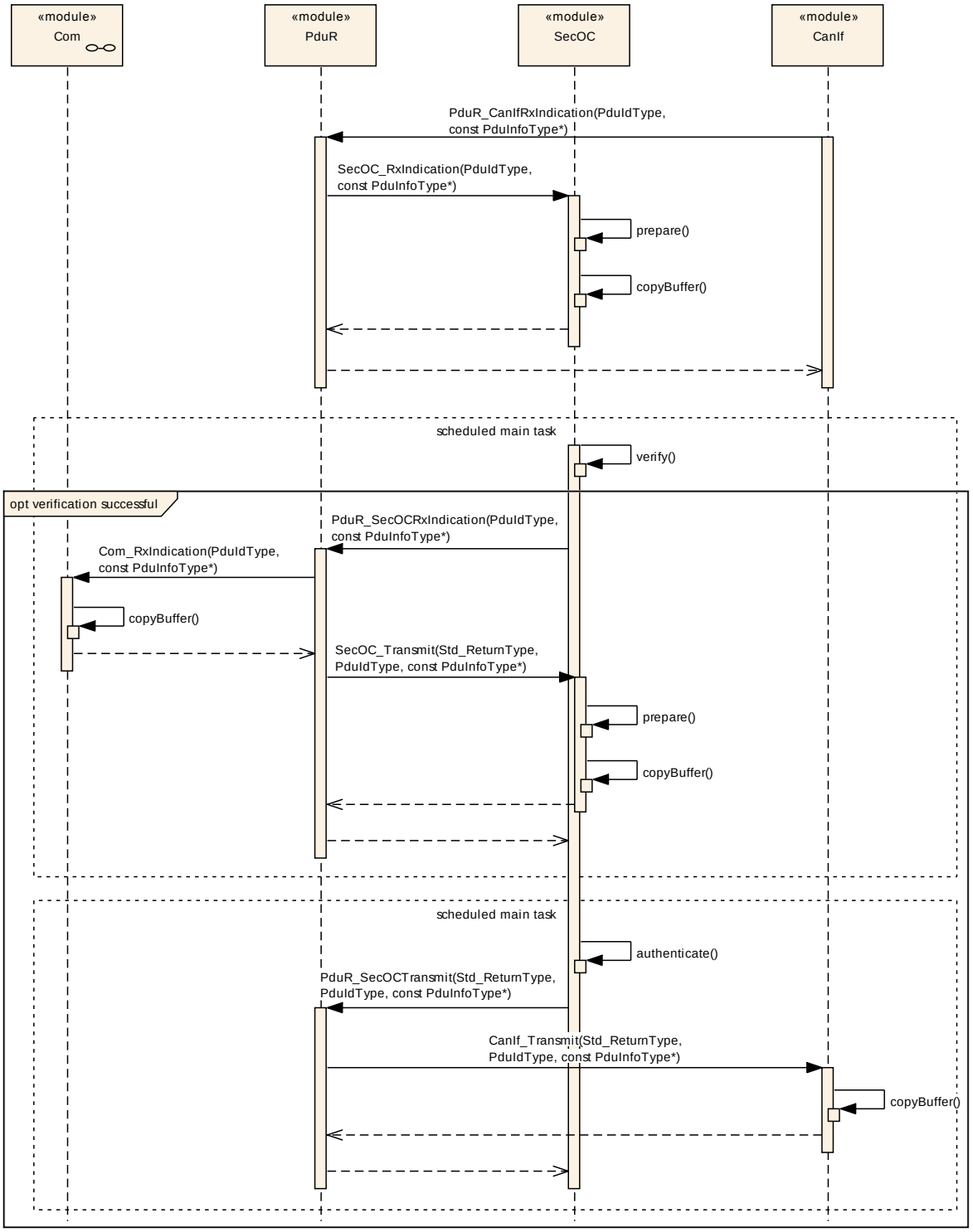

Figure 14 Verification and authentication in a gateway situation

10 Configuration specification

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in the Chapters below.

10.1.1 Variants

Currently three configuration variants for the AUTOSAR SecOC module are defined.

10.1.1.1 VARIANT-PRE-COMPILE

[SWS_SecOC_00143] ⌈VARIANT-PRE-COMPILE only supports pre-compile configurable parameters. Parameters below that are marked as Pre-compile configurable shall be configurable in a pre-compile manner, for example as #defines. A VARIANT-PRE-COMPILE module is most likely delivered as source code.⌋ (SRS_BSW_00345, SRS_BSW_00159)

Remark: Even though the module is delivered as source code, the implementation might use techniques similar to link time, i.e. table driven configuration.

10.1.1.2 VARIANT-LINK-TIME

[SWS_SecOC_00144] ⌈VARIANT-LINK-TIME includes mainly link-time and some pre-compile configurable parameters. All parameters defined below as link-time configurable shall be configurable at link time for example by linking a special configured parameter object file. A VARIANT-LINK-TIME module is most likely delivered as object code.⌋ (SRS_BSW_00159, SRS_BSW_00344)

10.1.1.3 VARIANT-POST-BUILD

[SWS_SecOC_00145] ⌈VARIANT-POST-BUILD includes post-build-time, link-time and some pre-compile configurable parameters. All parameters defined below as post build configurable shall be configurable post build for example by flashing configuration data. A VARIANT-POST-BUILD configurable module is most likely delivered as object code.⌋ (SRS_BSW_00404)

10.2 Containers and configuration parameters

For an overview of the AUTOSAR SecOC module's configuration, see Figure 15.

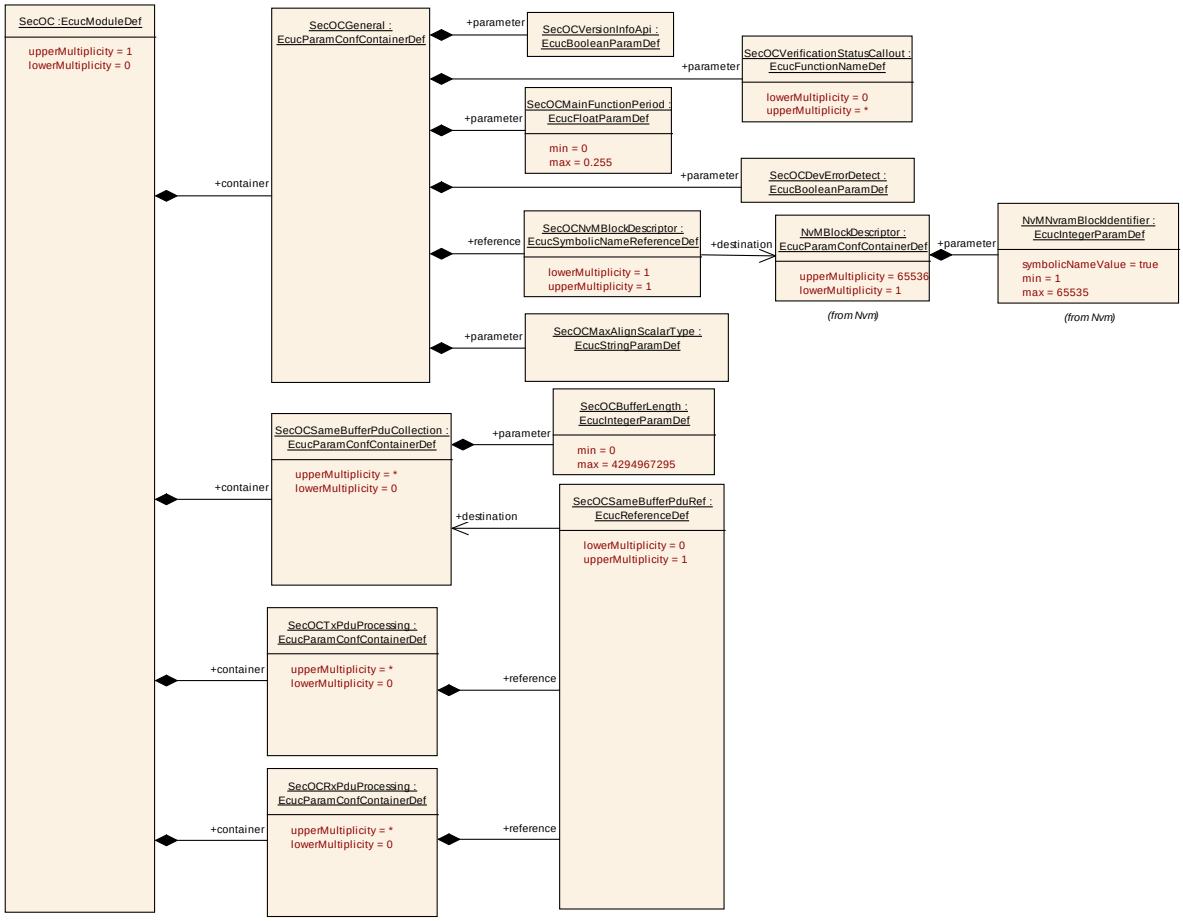

Figure 15: The AUTOSAR SecOC module's Configuration Overview

10.2.1 SecOC

SWS Item	**ECUC_SecOC_00001 :**
Module Name	*SecOC*
Module Description	Configuration of the SecOC (SecureOnboardCommunication) module.
Post-Build Variant Support	true

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SecOCGeneral	1	Contains the general configuration parameters of the SecOC module.
SecOCRxPduProcessing	0..*	Contains the parameters to configure the RxPdus to be verified by the SecOC module.
SecOCSameBufferPduCollection	0..*	SecOCBuffer configuration that may be used by a collection of Pdus.
SecOCTxPduProcessing	0..*	Contains the parameters to configure the TxPdus to be secured by the SecOC module.

10.2.2 SecOCGeneral

SWS Item	**ECUC_SecOC_00002 :**
Container Name	SecOCGeneral
Description	Contains the general configuration parameters of the SecOC module.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00007 :**		
Name	SecOCDevErrorDetect		
Description	Switches the Default Error Tracer (Det) detection and notification ON or OFF. • true: enabled (ON). • false: disabled (OFF).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00005 :**		
Name	SecOCMainFunctionPeriod		
Description	Allows to configure the time for the MainFunction (as float in seconds).		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 0.255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SecOC_00047 :**		
Name	SecOCMaxAlignScalarType		
Description	The scalar type which has the maximum alignment restrictions on the given platform. This type can be e.g. uint8, uint16 or uint32.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00004 :**		
Name	SecOCVerificationStatusCallout		
Description	Entry address of the customer specific call out routine which shall be invoked in case of a verification attempt.		
Multiplicity	0..*		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00003 :**		
Name	SecOCVersionInfoApi		
Description	If true the SecOC_GetVersionInfo API is available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00006 :**		
Name	SecOCNvMBlockDescriptor		
Description	Reference to NVRAM block containing the none volatile data. If this parameter is not configured it means that no NVRAM is used at all.		
Multiplicity	1		
Type	Symbolic name reference to [NvMBlockDescriptor]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.3 SecOCSameBufferPduCollection

SWS Item	**ECUC_SecOC_00009 :**
Container Name	SecOCSameBufferPduCollection
Description	SecOCBuffer configuration that may be used by a collection of Pdus.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00008 :**		
Name	SecOCBufferLength		
Description	This parameter defines the Buffer in bytes that is used by the SecOC module.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.4 SecOCRxPduProcessing

SWS Item	**ECUC_SecOC_00011 :**
Container Name	SecOCRxPduProcessing
Description	Contains the parameters to configure the RxPdus to be verified by the SecOC module.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00034 :**		
Name	SecOCAuthInfoTxLength		
Description	This parameter defines the length in bits of the authentication code to be included in the payload of the Secured I-PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00030 :**		
Name	SecOCDataId		
Description	This parameter defines a unique numerical identifier for the Secured I-PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00037 :**		
Name	SecOCFreshnessCounterSyncAttempts		
Description	This parameter defines the number of Freshness Counter re-synchronization attempts when a verification failed for a Secured I-PDU. If the value is zero, there will be no additional verification attempt to synchronize with a potentially better fitting Freshness Counter value. This parameter is only applicable if SecOCUseFreshnessTimestamp is FALSE.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00033 :**		
Name	SecOCFreshnessTimestampTimePeriodFactor		
Description	This parameter defines a factor that specifies the time period for the Freshness Timestamp. It holds a multiplication factor that specifies the concrete meaning of a Freshness Timestamp increment by one on basis of microseconds.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: SecOCUseFreshnessTimestamp == true		

SWS Item	**ECUC_SecOC_00038 :**		
Name	SecOCFreshnessValueId		
Description	This parameter defines the Id of the Freshness Value. The Freshness Value might be a normal counter or a time value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00031 :**
Name	SecOCFreshnessValueLength
Description	This parameter defines the complete length in bits of the Freshness Value. As long as the key doesn't change the counter shall not overflow. The length of the counter shall be determined based on the expected life time of the corresponding key and frequency of usage of the counter.
Multiplicity	1
Type	EcucIntegerParamDef

Range	0 .. 64		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00032 :**		
Name	SecOCFreshnessValueTxLength		
Description	This parameter defines the length in bits of the Freshness Value to be included in the payload of the Secured I-PDU. This length is specific to the least significant bits of the complete Freshness Counter. If the parameter is 0 no Freshness Value is included in the Secured I-PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 64		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: SecOCFreshnessCounterTxLength ≤ SecOCFreshnessCounterLength		

SWS Item	**ECUC_SecOC_00035 :**		
Name	SecOCKeyId		
Description	This parameter specifies the local Key identifier of the stored Key used to generate or verify a MAC.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00040 :**		
Name	SecOCRxAcceptanceWindow		
Description	This parameter defines the maximum allowed deviation in seconds from the expected timestamp for which a Secured I-PDU is still deemed authentic. This parameter is only applicable if SecOCUseFreshnessTimestamp is TRUE.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

	dependency: SecOCUseFreshnessTimestamp == true

SWS Item	**ECUC_SecOC_00039 :**		
Name	SecOCSecondaryFreshnessValueId		
Description	This parameter defines the Id of the Secondary Freshness Value. The Secondary Freshness Value might be a normal counter or a time value.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00036 :**		
Name	SecOCUseFreshnessTimestamp		
Description	This parameter specifies whether the Freshness Value is generated through individual Freshness Counters or by a Timestamps. The value is set to TRUE when Timestamps are used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00046 :**		
Name	SecOCVerificationStatusPropagationMode		
Description	This parameter is used to describe the propagation of the status of each verification attempt from the SecOC module to SWCs.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	BOTH		Both “True” and “False” AuthenticationStatus is propagated to SWC
	FAILURE_ONLY		Only “False” AuthenticationStatus is propagated to SWC
	NONE		No AuthenticationStatus is propagated to SWC
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00048 :**
Name	SecOCRxAuthServiceConfigRef
Description	This choice container is used to define which module and which service is used for verification.
Multiplicity	1
Type	Choice reference to [CalMacVerifyConfig , CalSignatureVerifyConfig , CsmMacVerifyConfig , CsmSignatureVerifyConfig]
Post-Build Variant Value	false
Scope / Dependency	

SWS Item	**ECUC_SecOC_00049 :**		
Name	SecOCSameBufferPduRef		
Description	This reference is used to collect Pdus that are using the same SecOC buffer.		
Multiplicity	0..1		
Type	Reference to [SecOCSameBufferPduCollection]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
SecOCRxAuthenticPduLayer	1	This container specifies the Pdu that is transmitted by the SecOC module to the PduR after the Mac was verified.
SecOCRxSecuredPduLayer	1	This container specifies the Pdu that is received by the SecOC module from the PduR. For this Pdu the Mac verification is provided.

10.2.5 SecOCRxSecuredPduLayer

SWS Item	**ECUC_SecOC_00041 :**
Container Name	SecOCRxSecuredPduLayer
Description	This container specifies the Pdu that is received by the SecOC module from the PduR. For this Pdu the Mac verification is provided.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00043 :**	
Name	SecOCRxSecuredLayerPduId	
Description	PDU identifier assigned by SecOC module. Used by PduR for SecOC_PduRRxIndication.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	--	
Post-Build Variant Value	false	

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

SWS Item	**ECUC_SecOC_00042 :**		
Name	SecOCRxSecuredLayerPduRef		
Description	Reference to the global Pdu.		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.6 SecOCRxAuthenticPduLayer

SWS Item	**ECUC_SecOC_00044 :**
Container Name	SecOCRxAuthenticPduLayer
Description	This container specifies the Pdu that is transmitted by the SecOC module to the PduR after the Mac was verified.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00045 :**		
Name	SecOCRxAuthenticLayerPduRef		
Description	Reference to the global Pdu.		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.7 SecOCTxPduProcessing

SWS Item	**ECUC_SecOC_00012 :**
Container Name	SecOCTxPduProcessing
Description	Contains the parameters to configure the TxPdus to be secured by the SecOC module.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00018 :**
Name	SecOCAuthInfoTxLength

Description	This parameter defines the length in bits of the authentication code to be included in the payload of the Secured I-PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00022 :**		
Name	SecOCAuthenticationRetries		
Description	This parameter defines the additional number of authentication attempts that are to be carried out when the generation of the authentication information failed for a given Secured I-PDU. If zero is set than only one authentication attempt is done.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00014 :**		
Name	SecOCDataId		
Description	This parameter defines a unique numerical identifier for the Secured I-PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00017 :**		
Name	SecOCFreshnessTimestampTimePeriodFactor		
Description	This parameter defines a factor that specifies the time period for the Freshness Timestamp. It holds a multiplication factor that specifies the concrete meaning of a Freshness Timestamp increment by one on basis of microseconds.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD

Scope / Dependency	scope: local dependency: SecOCUseFreshnessTimestamp == true

SWS Item	**ECUC_SecOC_00021 :**		
Name	SecOCFreshnessValueId		
Description	This parameter defines the Id of the Freshness Value. The Freshness Value might be a normal counter or a time value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00015 :**		
Name	SecOCFreshnessValueLength		
Description	This parameter defines the complete length in bits of the Freshness Value. As long as the key doesn't change the counter shall not overflow. The length of the counter shall be determined based on the expected life time of the corresponding key and frequency of usage of the counter.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 64		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00016 :**		
Name	SecOCFreshnessValueTxLength		
Description	This parameter defines the length in bits of the Freshness Value to be included in the payload of the Secured I-PDU. This length is specific to the least significant bits of the complete Freshness Counter. If the parameter is 0 no Freshness Value is included in the Secured I-PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 64		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: SecOCFreshnessCounterTxLength ≤ SecOCFreshnessCounterLength		

SWS Item	**ECUC_SecOC_00019 :**
Name	SecOCKeyId
Description	This parameter specifies the local Key identifier of the stored Key used to generate or verify a MAC.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00020 :**		
Name	SecOCUseFreshnessTimestamp		
Description	This parameter specifies whether the Freshness Value is generated through individual Freshness Counters or by a Timestamps. The value is set to TRUE when Timestamps are used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00010 :**		
Name	SecOCSameBufferPduRef		
Description	This reference is used to collect Pdus that are using the same SecOC buffer.		
Multiplicity	0..1		
Type	Reference to [SecOCSameBufferPduCollection]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00013 :**
Name	SecOCTxAuthServiceConfigRef
Description	This choice container is used to define which module and which service is used for authentication.
Multiplicity	1
Type	Choice reference to [CalMacGenerateConfig , CalSignatureGenerateConfig , CsmMacGenerateConfig , CsmSignatureGenerateConfig]
Post-Build Variant Value	false
Scope / Dependency	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SecOCTxAuthenticPduLayer	1	This container specifies the Pdu that is received by the SecOC module from the PduR. For this Pdu the Mac generation is provided.

SecOCTxSecuredPduLayer	1	This container specifies the Pdu that is transmitted by the SecOC module to the PduR after the Mac was generated.

10.2.8 SecOCTxAuthenticPduLayer

SWS Item	**ECUC_SecOC_00023 :**
Container Name	SecOCTxAuthenticPduLayer
Description	This container specifies the Pdu that is received by the SecOC module from the PduR. For this Pdu the Mac generation is provided.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00026 :**		
Name	SecOCTxAuthenticLayerPduId		
Description	PDU identifier assigned by SecOC module. Used by PduR for SecOC_PduRTransmit.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00025 :**		
Name	SecOCTxAuthenticLayerPduRef		
Description	Reference to the global Pdu.		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.9 SecOCTxSecuredPduLayer

SWS Item	**ECUC_SecOC_00024 :**
Container Name	SecOCTxSecuredPduLayer
Description	This container specifies the Pdu that is transmitted by the SecOC module to the PduR after the Mac was generated.
Configuration Parameters	

SWS Item	**ECUC_SecOC_00028 :**
Name	SecOCTxSecuredLayerPduId
Description	PDU identifier assigned by SecOC module. Used by PduR for confirmation

	(SecOC_PduRTxConfirmation) and for TriggerTransmit.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SecOC_00027 :**		
Name	SecOCTxSecuredLayerPduRef		
Description	Reference to the global Pdu.		
Multiplicity	1		
Type	Reference to [Pdu]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.3 Configuration Rules

SecOCFreshnessValueId and SecOCSecondaryFreshnessValueId are used to unambiguously identify the related I-PDU.

[SWS_SecOC_00147] ⌈The parameter values for SecOCFreshnessValueId and SecOCSecondaryFreshnessValueId shall be unique.
⌋

10.4 Published Information

For details, refer to the chapter 10.3 “Published Information” in SWS_BSWGeneral.

A Not applicable requirements

[SWS_SecOC_00999]⌈These requirements are not applicable to this specification.⌋(SRS_BSW_00004, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00158, SRS_BSW_00160, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00172, SRS_BSW_00300, SRS_BSW_00302, SRS_BSW_00304, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00310, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00318, SRS_BSW_00321, SRS_BSW_00325, SRS_BSW_00327, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00346, SRS_BSW_00347, SRS_BSW_00360, SRS_BSW_00361, SRS_BSW_00371, SRS_BSW_00374, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00405, SRS_BSW_00406, SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00412, SRS_BSW_00413, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00439, SRS_BSW_00440, SRS_BSW_00441, SRS_BSW_00442, SRS_BSW_00447, SRS_BSW_00448, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00458, SRS_BSW_00459, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00464, SRS_BSW_00465, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472)