MICROSAR CPL

Technical Reference

Version 1.2

Authors	Markus Schneider
Status	Released

Document Information

History

Author	Date	Version	Remarks
Markus Schneider	2014-04-09	1.00.00	Initial Version of MICROSAR Cpl
Markus Schneider	2014-10-14	1.01.00	Added AES-128 decryption/encryption Added AES-CTR DRBG (RNG) Adapted Configuration chapter
Markus Schneider	2014-12-15	1.01.01	Correction of buffer size in chapter 6.3.3
Markus Schneider	2014-12-19	1.02.00	Added RSA SHA256 Verify

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_CryptoAbstractionLibrary.pdf	1.2.0
[2]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[3]	AUTOSAR	AUTOSAR_SWS_RTE.pdf	3.2.0
[4]	NIST	Special Publication 800-38B: "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication"	May 2005
[5]	FIPS	FIPS PUB 186-2 “Digital Signature Standard (DSS)”	January 2000
[6]	Vector	MICROSAR Crypto Abstraction Library - Technical Reference	2.1

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

Caution
This symbol calls your attention to warnings.

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.0	Initial version
1.1	Added service AES decrypt/encrypt and AES-CTR DRBG
1.2	Added service RSA SHA256 Signature Verification

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the MICROSAR module CPL as specified in [1].

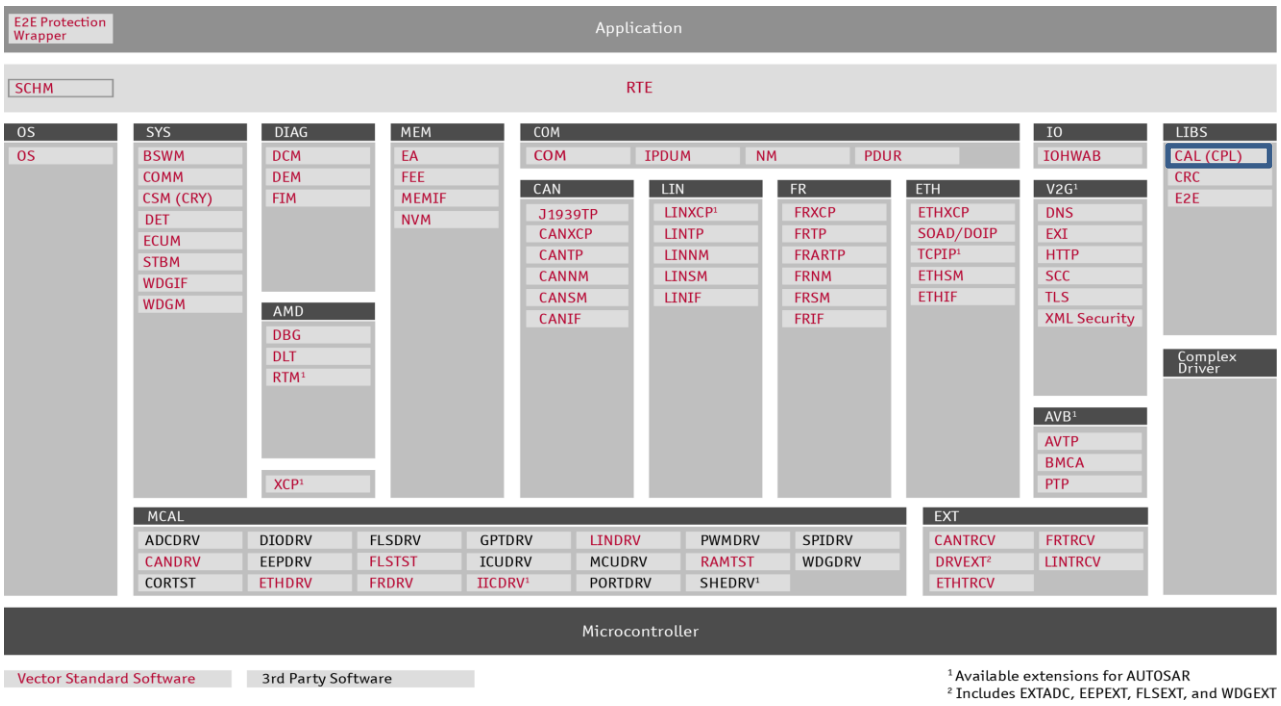
Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	CPL_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CPL_MODULE_ID	206 decimal (according to ref. [2])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

The Cryptographic Primitive Library (CPL) provides cryptographic algorithms for the MICROSAR Crypto Abstraction Library (CAL). This document describes mainly the configuration settings of the specific CPL services. Please refer to the 'MICROSAR Crypto Abstraction Library - Technical Reference' [6] on the configuration of the CAL.

2.1 Architecture Overview

The figure shows the interfaces to adjacent modules of the CPL (CAL). These interfaces are described in chapter 5.



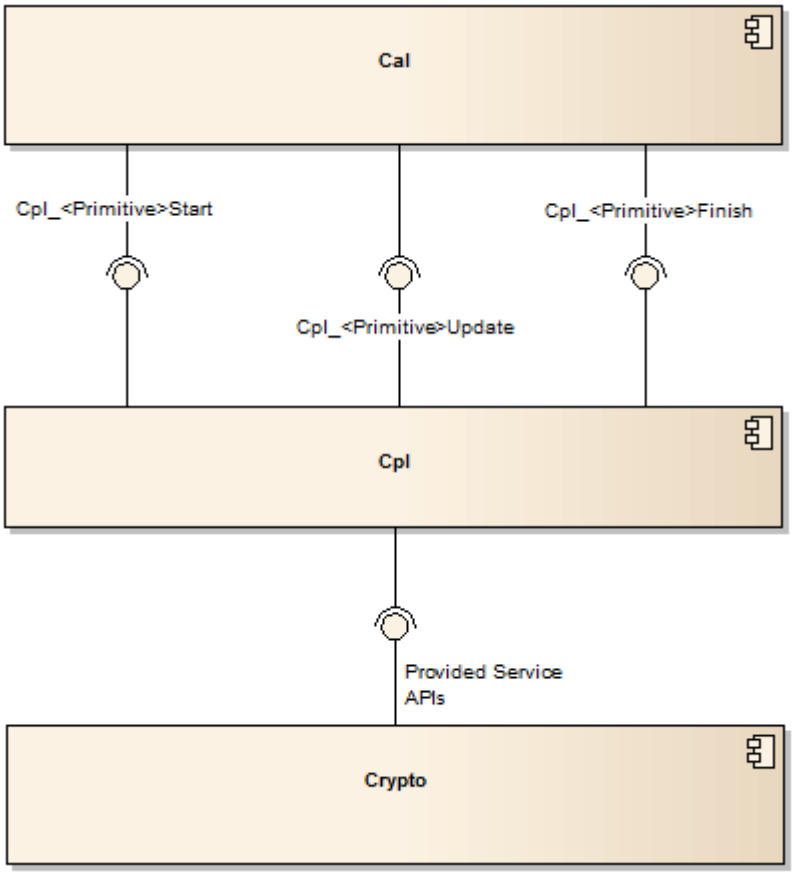


Figure 2-1 Interfaces to adjacent modules of the CPL

3 Functional Description

3.1 Features

The following sub-chapters describe the cryptographic primitives which can be included in your CPL delivery.

- AES-128 decrypt / encrypt
- AES-CMAC
- Random Number Generator (FIPS186-2)
- Random Number Generator (AES-CTR DRBG, NIST SP 800-90A)
- RSA SHA256 signature verification

3.2 AES-128 decrypt / encrypt

The CPL package can provide AES-128 modules to encrypt and decrypt data streams using the Advanced Encryption Standard. The key length of the encryption and decryption module is 128 bit.

3.2.1 Initialization

The initialization of the workspaces used by the AES modules is done in the specific start function.

3.2.2 Variants

The current implementation supports the Electronic Code Book Mode (ECB Mode) and the Cipher Block Chaining Mode (CBC Mode). The supported padding mode is PKCS#5.

Initialization Vector
Please note that the CBC mode requires an initialization vector whereas the ECB mode is accepting a NULL pointer.

3.3 AES-CMAC

The AES-CMAC is a cipher based ‘Message Authentication Code’ used to verify a message and provide integrity and authenticity on a message.

The algorithm was specified by the ‘National Institute of Standards and Technology’ (NIST) in the publication ‘Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication” [4]. The provided implementation fulfills this NIST specification.

A typical use-case of the AES-CMAC is shown in Figure 3-1 where a sender node transmits a message with an authentication code over a bus system to a receiver node.

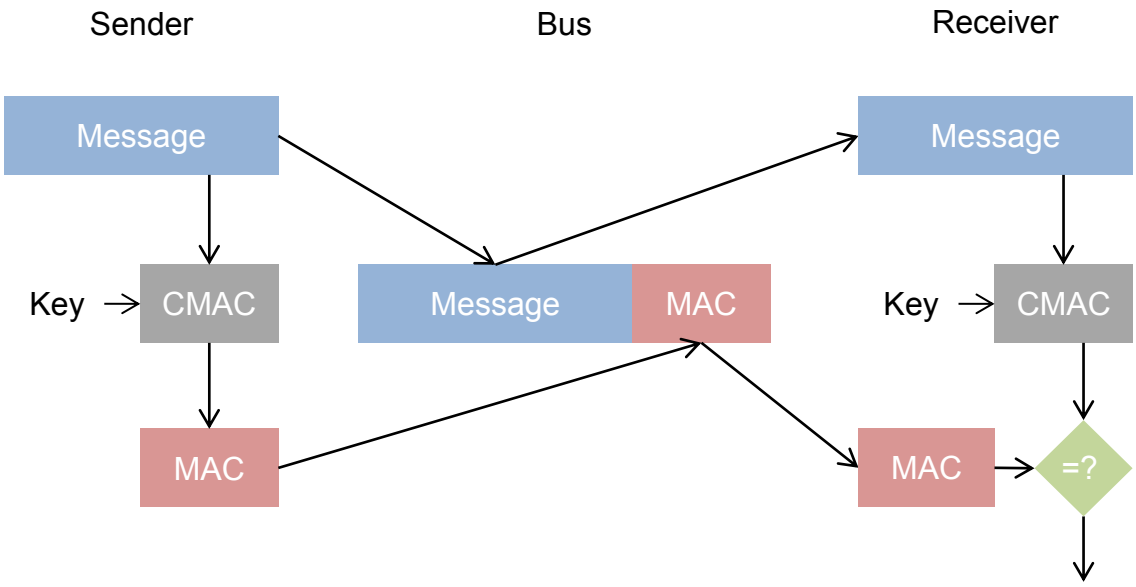


Figure 3-1 Typical MAC use-case

The sender node is using the AES-CMAC to create a MAC from the message and a shared secret key. The message is send along with the MAC over a bus system to the receiver node. The receiver also calculates a MAC over the message and checks the result against the received MAC to ensure the authenticity of the message.

3.3.1 Initialization

The initialization of the AES-CMAC service is done by calling the start function.

3.4 Random Number Generator (FIPS186-2)

The CPL provides an implemented random number generator (RNG) realized as a pseudo random number generator according to the FIPS PUB 186-2 [5] specification. The RNG has to be initialized with a seed which has to be a ‘true’ random number as the quality of randomness depends on the randomness of the seed input values since the seed provides the entropy for the whole random number generation.

3.4.1 Initialization

According to the CAL specification [1] no initialization of the CPL is needed.

However the service Cpl_Fips186Generate() prerequisites the initialization of the workspace, which includes the seed, by calling the Cpl_Fips186Seed<action>() APIs. As the FIPS186-2 standard requires at least a 20 byte random seed input, the implementation of the algorithm rejects requests with smaller seed values.

3.4.2 Variants

For development, testing purposes and special use-cases the pseudo random number generator provides the option to deactivate the save state. Disabling this feature will produce the same result for each call of Cal_RandomGenerate until the seed is updated

by the Cal_RandomSeed service. Using the seed update function more than once will stir up the seed value.

Using the save state has an impact on the seed on each call of the random generate API.

In default the save state should be activated as it is not needed to update the seed before every call of Cal_RandomGenerate to provide non-deterministic values. Configuration of this variant is done in the Cpl_Cfg.c. For further information on the configuration see 6.2.

3.5 Random Number Generator (AES-CTR DRBG, NIST SP 800-90A)

The NIST SP 800-90A specifies a mechanism for the generation of random bits using a deterministic block cipher algorithm. The CPL can use the AES-128 algorithm to implement a random number generator. Due to the CAL API design given by AUTOSAR the implementation has some limitations which can be found in 3.5.3.

3.5.1 Initialization

The initialization of the workspace used by the AES-CTR DRBG is done in the Seed functions. The Cpl_AesCtrDrbgGenerate() function prerequisites the initialization of the workspace by calling the Cpl_AesCtrDrbgSeed<action>() APIs.

3.5.2 Variants

The AES-CTR DRBG supports the use of AES-128 as block cipher algorithm.

There is no configurable variant.

3.5.3 Limitations

Due to the AUTOSAR design the nonce cannot be passed to the DRBG algorithm therefore the nonce is a counter value which will be incremented on every seed update call.

3.6 RSA SHA256 signature verification

The CPL may provide a signature verification service with the RSA SHA256 in the PKCS #1 1.5 variant.

3.6.1 Initialization

The initialization of the workspace used by the RSA SHA256 Verification is done by calling the start function. The mechanism of passing the key is slightly different from other CPL services. As the CAL interface does only provide a single key input parameter the implementation has to use the Cpl_RsaSha256KeyType instead of the Cal_AsymPublicKeyType. An example of the initialization can be found in 7.4.

3.6.2 Variants

There are no configurable variants available after the point of delivery.

The following key lengths can be delivered:

- 512 bit
- 1024 bit
- 1536 bit
- 2048 bit

3.7 Error Handling

The module does not provide any error detections according to AUTOSAR.

3.7.1 Development Error Reporting

The CAL as well as the CPL are not performing any error check at development level and also do not use the DET. Therefore there are no configuration settings necessary for error reporting.

3.7.2 Production Code Error Reporting

The module does not generate production code error reporting.

4 Integration

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

4.1 Scope of Delivery

The delivery of the CPL may contain the files described in chapters 4.1.1 and 4.1.2 depending on your order:

4.1.1 Static Files[1]

File Name	Source Code Delivery	Library Delivery	Description
Cpl.h	■		Header file of the CPL.
Cpl.lib		■	Library file of the CPL.
Cpl_AesCmac.h	■		Header file of AES-CMAC.
Cpl_AesCtrDrbg.h	■		Header file of AES-CTR DRBG.
Cpl_AesDecrypt128.h	■		Header file of AES-128 decrypt.
Cpl_AesEncrypt128.h	■		Header file of AES-128 encrypt.
Cpl_Fips186.h	■		Header file of Random Number Generator (FIPS 186-2).
Cpl_RsaSha256Verify.h	■		Header file of RSA SHA256 Signature Verifciaton.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the DaVinci Configurator 5. Refer to chapter 6.2 for details.

File Name	Description
Cpl_Cfg.h	This is the configuration header file.
Cpl_Cfg.c	This is the configuration source file.

Table 4-2 Generated files

[1] The file names and/or extensions may differ from your delivery

4.2 Include Structure

Figure 4-1 shows the include structure of the CPL. Some includes are optional and depend on the configuration. `Cpl_<Primitve>.h` stands for every used cryptographic primitive.

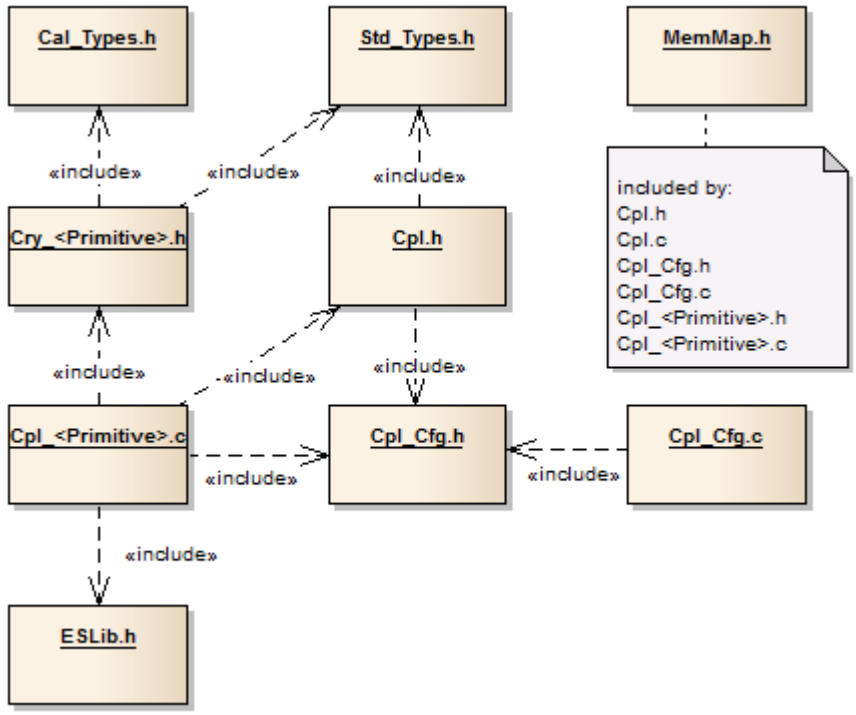


Figure 4-1 Include structure

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table (Table 4-3) contains the memory section names and the compiler abstraction definitions of the CPL and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	CPL_CODE	CPL_APPL_DATA	CPL_CONST
CPL_START_SEC_CODE CPL_STOP_SEC_CODE	■	■	
CPL_START_SEC_CONST_8BIT CPL_STOP_SEC_CONST_8BIT			■
CPL_START_SEC_CONST_UNSPECIFIED CPL_STOP_SEC_CONST_UNSPECIFIED			■

Table 4-3 Compiler abstraction and memory mapping

4.4 Critical Sections

The current implementation of the CPL module does not have any critical section.

5 API Description

5.1 Interfaces Overview

For an interfaces overview please see Figure 2-1.

5.2 Services provided by CPL

5.2.1 Cpl_AesCmac128VerifyStart

Prototype	
`Cal_ReturnType` **`Cpl_AESCMac128VerifyStart`** `(const void *cfgPtr, Cal_MacVerifyCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
keyPtr	Holds a pointer to the key necessary for the MAC verification.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function initializes the workspace needed for the CMAC verification.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-1 Cpl_AESCMac128VerifyStart

5.2.2 Cpl_AesCmac128VerifyUpdate

Prototype	
`Cal_ReturnType` **`Cpl_AESCMac128VerifyUpdate`** `(const void *cfgPtr, Cal_MacVerifyCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
dataPtr	Holds a pointer to the data for which a MAC shall be verified.

dataLength	Contains the number of bytes for which the MAC shall be verified.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function updates the CMAC verification.	
Particularities and Limitations	
The dataLength has to be 16 byte unless it isn't the last data block.	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-2 Cpl_AESCMac128VerifyUpdate

5.2.3 Cpl_AesCmac128VerifyFinish

Prototype	
`Cal_ReturnType` **`Cpl_AESCMac128VerifyFinish`** `(const void *cfgPtr, Cal_MacVerifyCtxBufType contextBuffer, const uint8 *MacPtr, uint32 MacLength, Cal_VerifyResultType *resultPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
MacPtr	Holds a pointer to the memory location which will hold the MAC to verify.
MacLength	Holds the length of the MAC to be verified.
resultPtr	Holds a pointer to the memory location which will hold the result of the MAC verification.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function finishs the CMAC generation and verify it with the given MacPtr.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-3 Cpl_AESCMac128VerifyFinish

5.2.4 Cpl_AESCMac128GenerateStart

Prototype	
`Cal_ReturnType` **`Cpl_AESCMac128GenerateStart`** `(const void *cfgPtr, Cal_MacGenerateCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
keyPtr	Holds a pointer to the key necessary for the MAC generation.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function initializes the workspace needed for the CMac generation.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-4 Cpl_AESCMac128GenerateStart

5.2.5 Cpl_AESCMac128GenerateUpdate

Prototype	
`Cal_ReturnType` **`Cpl_AESCMac128GenerateUpdate`** `(const void *cfgPtr, Cal_MacGenerateCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
dataPtr	Holds a pointer to the data for which a MAC shall be computed.
dataLength	Contains the number of bytes for which the MAC shall be computed.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function updates the CMAC generation.	

Particularities and Limitations
The dataLength has to be 16 byte unless it isn't the last data block.
Pre-Conditions
Call Context
Called by CAL.

Table 5-5 Cpl_AESCMac128GenerateUpdate

5.2.6 Cpl_AESCMac128GenerateFinish

Prototype	
`Cal_ReturnType` **`Cpl_AESCMac128GenerateFinish`** `(const void *cfgPtr, Cal_MacGenerateCtxBufType contextBuffer, uint8 *resultPtr, uint32 *resultLengthPtr, boolean TruncationIsAllowed)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
resultPtr	Holds a pointer to the memory location which will hold the result of the MAC generation. If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated.
resultLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by resultPtr. On returning from this function the actual length of the computed MAC shall be stored.
TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: Truncation is allowed. FALSE: Truncation is not allowed.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed CAL_E_SMALL_BUFFER – The provided buffer is too small to store the result, and truncation was not allowed.
Functional Description	
This function finish the CMAC generation and verify it with the given MacPtr.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-6 Cpl_AESCMac128GenerateFinish

5.2.7 Cpl_AesCtrDrbgSeedStart

Prototype	
`Cal_ReturnType` **`Cpl_AesCtrDrbgSeedStart`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function initializes the workspace for the random seed service.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-7 Cpl_AesCtrDrbgSeedStart

5.2.8 Cpl_AesCtrDrbgSeedUpdate

Prototype	
`Cal_ReturnType` **`Cpl_AesCtrDrbgSeedUpdate`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer, const uint8 *seedPtr, uint32 seedLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
seedPtr	Holds a pointer to the seed for the random number generator.
seedLength	Contains the length of the seed in bytes.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function shall be used to feed a seed to the random number generator.	
Pre-Conditions	

Call Context
Called by CAL.

Table 5-8 Cpl_AesCtrDrbgSeedUpdate

5.2.9 Cpl_AesCtrDrbgSeedFinish

Prototype	
`Cal_ReturnType` **`Cpl_AesCtrDrbgSeedFinish`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function finalizes the random seed service.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-9 Cpl_AesCtrDrbgSeedFinish

5.2.10 Cpl_AesCtrDrbgGenerate

Prototype	
`Cal_ReturnType` **`Cpl_AesCtrDrbgGenerate`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer, uint8 *resultPtr, uint32 resultLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
resultPtr	Holds a pointer to the memory location which will hold the result of the random number generation. The memory location must have at least the size "resultLength".
resultLength	Holds the amount of random bytes which should be generated.

Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
Generates a random number according to the NIST800-90A specification.	
Particularities and Limitations	
none	
Pre-Conditions	
The context buffer must be the same as used for the call of the RandomSeed interface.	
Call Context	
Called by CAL.	

Table 5-10 Cpl_AesCtrDrbgGenerate

5.2.11 Cpl_AesDecrypt128Start

Prototype	
`Cal_ReturnType` **`Cpl_AesDecrypt128Start`** `(const void *cfgPtr, Cal_SymDecryptCtxBufType contextBuffer, Cal_SymKeyType keyPtr, uint8 InitVectorPtr, uint32 InitVectorLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
keyPtr	Holds a pointer to the key which has to be used during the symmetrical decryption operation.
InitVectorPtr	Holds a pointer to initialization vector which has to be used during the symmetrical decryption.
InitVectorLength	Holds the size of the initialization vector in bytes.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface initializes the AES 128 decryption service.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-11 Cpl_AesDecrypt128Start

5.2.12 Cpl_AesDecrypt128Update

Prototype	
`Cal_ReturnType` **`Cpl_AesDecrypt128Update`** `(const void *cfgPtr, Cal_SymDecryptCtxBufType contextBuffer, const uint8 * cipherTextPtr, uint32 cipherTextLength, uint8 * plainTextPtr, uint32 * plainTextLengthPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
cipherTextPtr	Holds a pointer to the data for which a decrypted text shall be computed.
cipherTextLength	Contains the number of bytes for which the decrypted text shall be computed.
plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
plainTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. When the request has finished, the actual length of the returned decrypted text shall be stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface is used to feed the AES 128 decryption service with the input data.	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-12 Cpl_AesDecrypt128Update

5.2.13 Cpl_AesDecrypt128Finish

Prototype	
`Cal_ReturnType` **`Cpl_AesDecrypt128Finish`** `(const void *cfgPtr, Cal_SymDecryptCtxBufType contextBuffer, uint8 * plainTextPtr, uint32 * plainTextLengthPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.

plainTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. When the request has finished, the actual length of the returned decrypted text is stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface is used to finish the AES decryption service.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-13 Cpl_AesDecrypt128Finish

5.2.14 Cpl_AesEncrypt128Start

Prototype
`Cal_ReturnType` **`Cpl_AesEncrypt128Start`** `(const void *cfgPtr, Cal_SymDecryptCtxBufType contextBuffer, Cal_SymKeyType keyPtr, uint8 InitVectorPtr, uint32 InitVectorLength)`
Parameter
cfgPtr — Pointer to the configuration.
contextBuffer — Holds the pointer to the buffer in which the context of this service can be stored.
keyPtr — Holds a pointer to the key which has to be used during the symmetrical encryption operation.
InitVectorPtr — Holds a pointer to initialization vector which has to be used during the symmetrical encryption.
InitVectorLength — Holds the size of the initialization vector in bytes.
Return Code
Cal_ReturnType — CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description
This interface initializes the AES 128 encryption service.
Particularities and Limitations
none
Pre-Conditions
Call Context

Called by CAL.

Table 5-14 Cpl_AesEncrypt128Start

5.2.15 Cpl_AesEncrypt128Update

Prototype	
`Cal_ReturnType Cpl_AesEncrypt128Update (const void *cfgPtr, Cal_SymDecryptCtxBufType contextBuffer, const uint8 * plainTextPtr, uint32 plainTextLength, uint8 * cipherTextPtr, uint32 * cipherTextLengthPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
plainTextPtr	Holds a pointer to the data for which a encrypted text shall be computed.
plainTextLength	Contains the number of bytes for which the encrypted text shall be computed.
cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
cipherTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. When the request has finished, the actual length of the returned encrypted text is stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface is used to feed the AES 128 encryption service with the input data.	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-15 Cpl_AesEncrypt128Update

5.2.16 Cpl_AesEncrypt128Finish

Prototype	
`Cal_ReturnType Cpl_AesEncrypt128Finish (const void *cfgPtr, Cal_SymDecryptCtxBufType contextBuffer, uint8 * cipherTextPtr, uint32 * cipherTextLengthPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
cipherTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. When the request has finished, the actual length of the returned encrypted text shall be stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface is used to finish the AES encryption service.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-16 Cpl_AesEncrypt128Finish

5.2.17 Cpl_FIPS186SeedStart

Prototype
`Cal_ReturnType` **`Cpl_FIPS186SeedStart`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer)`
Parameter
cfgPtr: Pointer to the configuration.
contextBuffer: Holds the pointer to the buffer in which the context of this service can be stored.
Return Code
Cal_ReturnType: CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description
This function initializes the workspace for the random seed service.
Particularities and Limitations
none
Pre-Conditions
Call Context
Called by CAL.

Table 5-17 Cpl_FIPS186SeedStart

5.2.18 Cpl_FIPS186SeedUpdate

Prototype	
`Cal_ReturnType` **`Cpl_FIPS186SeedUpdate`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer, const uint8 *seedPtr, uint32 seedLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
seedPtr	Holds a pointer to the seed for the random number generator.
seedLength	Contains the length of the seed in bytes.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function shall be used to feed a seed to the random number generator.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-18 Cpl_FIPS186SeedUpdate

5.2.19 Cpl_FIPS186SeedFinish

Prototype	
`Cal_ReturnType` **`Cpl_FIPS186SeedFinish`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This function does actually nothing but is needed for the service configuration function pointer.	
Particularities and Limitations	
none	

Pre-Conditions
Call Context
Called by CAL.

Table 5-19 Cpl_FIPS186SeedFinish

5.2.20 Cpl_FIPS186Generate

Prototype	
`Cal_ReturnType` **`Cpl_FIPS186Generate`** `(const void *cfgPtr, Cal_RandomCtxBufType contextBuffer, uint8 *resultPtr, uint32 resultLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
Generates a random number according to the FIPS186-2 specification.	
Particularities and Limitations	
none	
Pre-Conditions	
The context buffer must be the same that has been used for the call of the FIPS186 RandomSeed interface.	
Call Context	
Called by CAL.	

Table 5-20 Cpl_FIPS186Generate

5.2.21 Cpl_RsaSha256VerifyStart

Prototype	
`Cal_ReturnType` **`Cpl_RsaSha256VerifyStart`** `(const void *cfgPtr, Cal_SignatureVerifyCtxBufType contextBuffer, const Cal_AsymPublicKeyType *keyPtr )`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

keyPtr	Holds a pointer to the key buffer necessary for the signature verification. The key buffer has to be of type Cpl_RsaSha256KeyType.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface initialize the RSA SHA256 signature verify service.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-21 Cpl_RsaSha256VerifyStart

5.2.22 Cpl_RsaSha256VerifyUpdate

Prototype	
`Cal_ReturnType` **`Cpl_RsaSha256VerifyUpdate`** `(const void *cfgPtr, Cal_SignatureVerifyCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
dataPtr	Holds a pointer to the message which shall be verified.
dataLength	Contains the length of the message to verify in bytes.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface is used to feed the RSA SHA256 signature verification service with the input data.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-22 Cpl_RsaSha256VerifyUpdate

5.2.23 Cpl_RsaSha256VerifyFinish

Prototype	
`Cal_ReturnType` **`Cpl_RsaSha256VerifyFinish`** `(const void *cfgPtr, Cal_SignatureVerifyCtxBufType contextBuffer, const uint8 *signaturePtr, uint32 signatureLength, Cal_VerifyResultType *resultPtr)`	
Parameter	
cfgPtr	Pointer to the configuration.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
signaturePtr	Holds a pointer to the memory location which holds the signature to be verified.
signatureLength	Holds the length of the Signature to be verified.
resultPtr	Holds a pointer to the memory location which will hold the result of the signature verification.
Return Code	
Cal_ReturnType	CAL_E_OK – Request successful CAL_E_NOT_OK – Request failed
Functional Description	
This interface is used to finish the RSA SHA256 signature verification service.	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
Called by CAL.	

Table 5-23 Cpl_RsaSha256VerifyFinish

6 Configuration

The CPL can be configured with the following tools:

> DaVinci Configurator 5

6.1 Configuration Variants

The CPL supports only the configuration variant

> VARIANT-PRE-COMPILE

6.2 Configuration with DaVinci Configurator 5

6.2.1 Common Properties

Attribute Name	Values Default value is typed bold	Description
Buffer Size Check	**STD_ON** STD_OFF	Enables the buffer size check on compile time.
Buffer Offset	**1**	This value describes the delta from the received buffer by the CAL and the provided one in bytes. The MICROSAR CAL is using one byte to save the internal state. For this reason the provided buffer is one byte smaller.

6.2.2 CplAesCtrDrbg Configuration

Attribute Name	Values Default value is typed bold	Description
Block Length	**CPL_SIZEOF_AES128_BLOCK**	Block length used by the CTR-DRBG
Key Length	**CPL_SIZEOF_AES128_KEY**	Key length used by the CTR-DRBG
Reseed Interval	**1000**	This value describes how much random numbers can be generated until the entropy is exceeded.
Reseed Use Nonce	**TRUE** FALSE	If this checkbox is enabled the reseed function is using the nonce (counter) as additional input.
Use Derivation Function	**TRUE** FALSE	When no source of full entropy is available usage of the derivation function is required.

Attribute Name	Values Default value is typed bold	Description
Private String	String	Private string used by the seed instantiation.
Private Sting Length	0.. 16	Length of the private string in bytes.

6.2.3 CplAesDecrypt128/ CplAesEncrypt128 Configuration

Attribute Name	Values Default value is typed bold	Description
AES Decrypt/Encrypt Block Mode	**CPL_AESBLOCKMODE_ECB** CPL_AESBLOCKMODE_CBC	Block chaining mode used for the decryption/encryption with AES
AES Decrypt/Encrypt Padding Mode	**CPL_AESPADDINGMODE_PKCS5**	Padding mode used by the AES decryption/encryption service

6.2.4 CplFips186 Configuration

Attribute Name	Values Default value is typed bold	Description
Save State	**TRUE** FALSE	Enable storing the RNG state. This shall always be TRUE. Except for debugging or testing reasons.

6.3 Configuration of the CAL

For information about the configuration in CAL please refer to the Technical Reference of the CAL [6].

CPL References
References in the CAL configuration module can be targeted to the specific CPL services. Most configuration options will be prefilled with the according data.

6.3.1 Configuration hints for AES-CMAC

Type	Description
Name	AES-CMAC
CAL classification	Cal_MacVerify Cal_MacGenerate
Files	Cpl_AesCmac.c Cpl_AesCmac.h
API naming scheme	Cpl_AesCmac
Buffer size needed[2]	292 byte
Key size	16 byte

Table 6-1 AES-CMAC

6.3.2 Configuration hints for AES-CTR DRBG

Type	Description
Name	AES-CTR DRBG
CAL classification	Cal_RandomSeed

Type	Description
	Cal_RandomGenerate
Files	Cpl_AesCtrDrbg.c Cpl_AesCtrDrbg.h
API naming scheme	Cpl_AesCtrDrbg
Buffer size needed[2]	321 byte

Table 6-2 AES-CTR DRBG

6.3.3 Configuration hints for AES-128 decryption/encryption

Type	Description
Name	AES-128 decryption/encryption
CAL classification	Cal_SymDecrypt Cal_SymEncrypt
Files	Cpl_AesDecrypt128.c Cpl_AesDecrypt128.h Cpl_AesEncrypt128.c Cpl_AesEncrypt128.h
API naming scheme	Cpl_AesDecrypt128 / Cpl_AesEncrypt128
Buffer size needed[2]	317 byte
Key size	16 byte

Table 6-3 AES-128 decryption/encryption

6.3.4 Configuration hints for RNG FIPS186-2

Type	Description
Name	RNG FIPS186-2
CAL classification	Cal_RandomSeed Cal_RandomGenerate
Files	Cpl_Fips186.c Cpl_Fips186.h
API naming scheme	Cpl_Fips186
Buffer size needed[2]	153 byte

Table 6-4 FIPS186

6.3.5 Configuration hints for RSA SHA256 Signature Verification

Type	Description
Name	RSA SHA256 Signature Verify
CAL classification	Cal_SignatureVerify
Files	Cpl_RsaSha256Verify.c Cpl_RsaSha256Verify.h

[2] This value may vary due to the hardware architecture design.

Type	Description
API naming scheme	Cpl_RsaSha256Verify
Buffer size needed[3]	1993 byte

Table 6-5 RSA SHA256 Verify

[3] This value may vary due to the hardware architecture design.

7 Usage

This chapter describes the usage of the CPL module. The provided examples presuppose the correct configuration within the CAL module.

7.1 MAC Verification

The following sample is using the MacVerification service to verify a `message` against a `mac`.

```
Cal_MacVerifyCtxBufType macBuffer;

Cal_VerifyResultType verifyResult;

uint8 message[64] = {0x6bu, 0xc1u, 0xbeu, 0xe2u, 0x2eu, 0x40u, 0x9fu, 0x96u, 0xe9u,
0x3du, 0x7eu, 0x11u, 0x73u, 0x93u, 0x17u, 0x2au, 0xaeu, 0x2du, 0x8au, 0x57u, 0x1eu,
0x03u, 0xacu, 0x9cu, 0x9eu, 0xb7u, 0x6fu, 0xacu, 0x45u, 0xafu, 0x8eu, 0x51u, 0x30u,
0xc8u, 0x1cu, 0x46u, 0xa3u, 0x5cu, 0xe4u, 0x11u, 0xe5u, 0xfbu, 0xc1u, 0x19u, 0x1au,
0x0au, 0x52u, 0xefu, 0xf6u, 0x9fu, 0x24u, 0x45u, 0xdfu, 0x4fu, 0x9bu, 0x17u, 0xadu,
0x2bu, 0x41u, 0x7bu, 0xe6u, 0x6cu, 0x37u, 0x10u};

uint8 remainingInput = sizeof(message);

uint8 mac[16] = {0x51u, 0xf0u, 0xbeu, 0xbfu, 0x7eu, 0x3bu, 0x9du, 0x92u, 0xfcu, 0x49u,
0x74u, 0x17u, 0x79u, 0x36u, 0x3cu, 0xfeu};

Cal_SymKeyType key = { 16u, {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c}};


if ( CAL_E_OK == Cal_MacVerifyStart( CalMacVerifyConfig,
                                     macBuffer,
                                     &key ) )
{
  while (remainingInput > 0u)
  {
    inputSize = (remainingInput >= 16u) ? (16u) : (remainingInput);
    if ( CAL_E_OK != Cal_MacVerifyUpdate( CalMacVerifyConfig,
                                          macBuffer,
                                          &message[(sizeof(message) - remainingInput)],
                                          inputSize ) )
    {
      ...
    }
    remainingInput -= inputSize;
  }
}


if ( 0u == remainingInput )
{
  if ( CAL_E_OK == Cal_MacVerifyFinish( CalMacVerifyConfig,
```

```
                                        macBuffer,
                                        mac,
                                        sizeof(mac),
                                        resultPtr ) )
  {
    if ( CAL_E_VER_NOT_OK == resultPtr )
    {
      /* Verification successful */
    }
  }
}
```

7.2 Symmetrical Encryption

This is an example which demonstrates the usage of the symmetrical encryption:

```
Cal_SymEncryptCtxBufType ctxBuffer;

Cal_SymKeyType key= {16u, { 0x00u, 0x01u, 0x02u, 0x03u, 0x04u, 0x05u, 0x06u, 0x07u,
0x08u, 0x09u, 0x0au, 0x0bu, 0x0cu, 0x0du, 0x0eu, 0x0fu}};


uint8 plainText[16] = { 0x00u, 0x11u, 0x22u, 0x33u, 0x44u, 0x55u, 0x66u, 0x77u, 0x88u,
0x99u, 0xaau, 0xbbu, 0xccu, 0xddu, 0xeeu, 0xffu };

uint8 cipherText[32];

uint32 cipherSize = sizeof(cipherText);


if ( CAL_E_OK != Cal_SymEncryptStart( CalSymEncryptConfig , ctxBuffer, &key, 0, 0 ) )

{}


if ( CAL_E_OK != Cal_SymEncryptUpdate( CalSymEncryptConfig, ctxBuffer, plainText, 16u,
cipherText, &cipherSize ) )

{}


if ( CAL_E_OK != Cal_SymEncryptFinish( CalSymEncryptConfig, ctxBuffer, &cipherText[16],
&cipherSize ) )

{}
```

Note
With the AES-128 in streaming mode the padding is always applied even if the plain text input size is equal to the block size of the AES.

7.3 Random Number Generator

In this example a seed value will be initialized and 10 byte random data generated.

```
Cal_RandomCtxBufType randomBuffer;
```

```
uint8 seed[20] = {0x20u, 0xDDu, 0x71u, 0xDDu, 0x89u, 0x19u, 0x99u, 0x8Cu, 0x85u, 0xF1u,
0x87u, 0xE3u, 0xCFu, 0xF4u, 0x10u, 0x57u, 0x84u, 0x1Cu, 0x1Cu, 0xABu};
uint8 randomOutput[10];

/* Initialize seed */
if ( CAL_E_OK != Cal_RandomSeedStart( CalRandomSeedConfig, randomBuffer ) )
{
}
if ( CAL_E_OK != Cal_RandomSeedUpdate( CalRandomSeedConfig, randomBuffer,
                                       seed, sizeof(seed) ) )
{
}
if ( CAL_E_OK != Cal_RandomSeedFinish( CalRandomSeedConfig, randomBuffer ) )
{
}
/* Generate a random sequence */
if ( CAL_E_OK != Cal_RandomGenerate( CalRandomGenerateConfig,
                                     randomBuffer,
                                     randomOutput,
                                     sizeof(randomOutput) ) )
{
}
```

7.4 Signature Verification

This example demonstrates the usage of the signature verification service with RSA SHA256.

```
  Cal_SignatureVerifyCtxBufType ctxBuffer;
  Cal_VerifyResultType result;

  Cpl_RsaSha256KeyType key;

  key.keyExponent = publicExponent;
  key.keyExponentLength = sizeof(publicExponent);
  key.keyModule = publicModulus;
  key.keyModuleLength = sizeof(publicModulus);

if ( CAL_E_OK != Cal_SignatureVerifyStart( CalSignatureVerifyConfig, ctxBuffer,
                                           (Cal_AsymPublicKeyType*)&key ) )
{
}
```

```
if ( CAL_E_OK != Cal_SignatureVerifyUpdate( CalSignatureVerifyConfig, ctxBuffer,
                                            message, sizeof(message) ) )
{
}

if ( CAL_E_OK != Cal_SignatureVerifyFinish( CalSignatureVerifyConfig, ctxBuffer,
                                            signature, sizeof(signature), result ) )
{
}

if ( CAL_E_VER_OK == result )
{
   /* Verification ok */
}
```

8 AUTOSAR Standard Compliance

8.1 Deviations

The current implementation does not have any deviations.

8.2 Additions/ Extensions

The current implementation does not have any extensions.

8.3 Limitations

8.3.1 Support of Cryptographic Services

Currently, only the cryptographic services are supported:

▶ AES-CMAC - Service for MAC Interface
▶ AES-128 Decrypt/Encrypt – Service for Symmetrical Interface
▶ RNG AES-CTR DRBG – Service for Random Interface
▶ RNG FIPS186-2 – Service for Random Interface
▶ RSA SHA256 Signature Verify – Service for Signature Interface

Table 8-1 Supported AUTOSAR standard conform features

The following cryptographic services are not supported yet:

▶ Service for Hash Interface
▶ Service for Symmetrical Block Interface
▶ Service for Asymmetrical Interface
▶ Service for Signature Interface (Generate)
▶ Service for Checksum Interface
▶ Service for Key Derivation Interface
▶ Service for Key Exchange Interface
▶ Service for Symmetrical Key Exchange Interface
▶ Service for Symmetrical Key Extract Interface
▶ Service for Asymmetrical Key Extract Interface

8.3.2 Tool supported configuration

Tool support is provided by the DaVinci Configurator 5.

9 Glossary and Abbreviations

9.1 Glossary

Term	Description
Cryptographic Primitive	An underlying cryptographic module or library

Table 9-1 Glossary

9.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CPL	Cryptographic Primitive Library
CAL	Cryptographic Abstraction Library
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SchM	Schedule Manager
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 9-2 Abbreviations

10 Contact

Visit our website for more information on

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

www.vector.com