VECTOR

MICROSAR CRYPTO

Technical Reference

CRYPTO LIBCV

Version 2.1.0

Authors	Markus Schneider, Matthias Weniger
Status	Released

Document Information

History

Author	Date	Version	Remarks
Schneider, Markus	2017-03-07	1.01.00	Initial creation of Technical Reference
Weniger, Matthias	2017-08-31	2.01.00	Rework Document Updated supported features 3.1 Added chapter 3.1.3.3 Added chapter 3.1.3.4 Added chapter 3.7 Added chapter 3.8 Added chapter 3.9 Added description of all used EXCLUSIVE AREAS in chapter 4.2 Updated for behavior changes 5.2.9

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_CryptoDriver.pdf	4.3.0
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.3.0
[3]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	4.3.0
[4]	HIS	2009-04-01 SHE Functional Specification v1.1 (rev439)	1.1

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.00	Initial beta release
1.01	Adaptions to the specification; several improvements and bugfixes
2.01	Safe BSW; Support of reuse of round-keys for AES CMAC ; several improvements and bugfixes

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4.3	
Supported Configuration Variants:	pre-compile	
Vendor ID:	CRYPTO_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CRYPTO_MODULE_ID	114 decimal (according to ref. [3])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The Crypto Driver (CRYPTO) is called by the Crypto Interface (CRYIF) and performs the specific cryptographic functionality. The CRYPTO specification [1] offers a superset of algorithms which can be extended by ‘custom algorithms’. This software-based Crypto Driver offers a subset of algorithms and features which is described in 3.1.

2.1 Architecture Overview

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

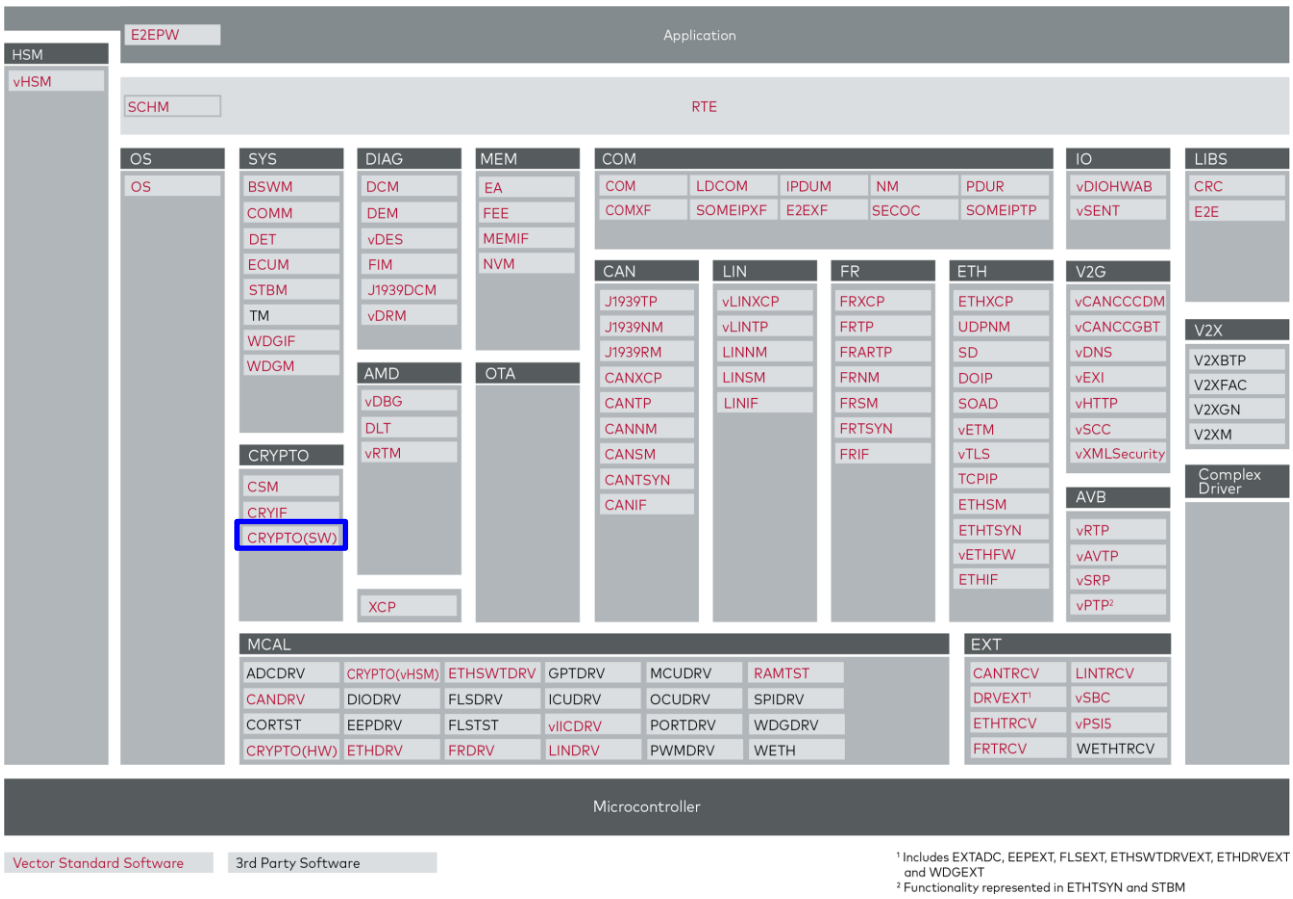

Figure 2-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the CRYPTO. These interfaces are described in chapter 5.

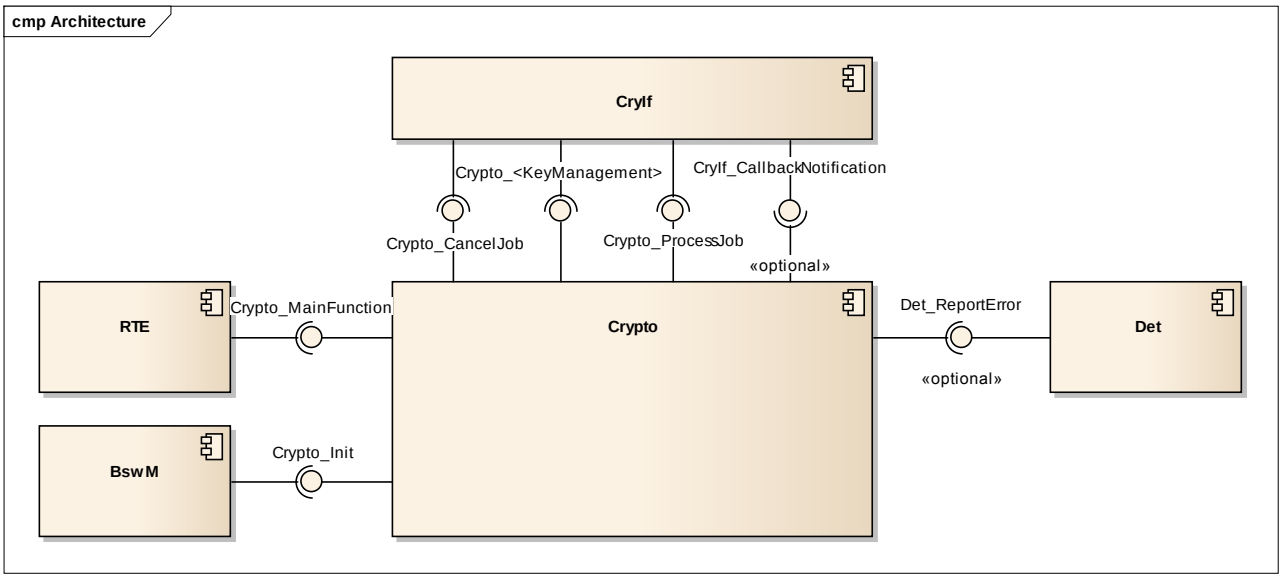

Figure 2-2 Interfaces to adjacent modules of the CRYPTO

3 Functional Description

3.1 Features

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

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

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further CRYPTO functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
AEAD: AES-GCM
Encrypt/Decrypt: AES-128 ECB/CBC
HASH: SHA-1; SHA-256; SHA-512
MAC: AES CMAC; SipHash
PRNG: FIPS 186-2
Signature: Ed25519
Key Derive: KDF in Counter Mode and KDF in Counter Mode with Appendix

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
Certificate handling
All algorithms not stated in Table 3-1
The Read/Write Access for Key Elements is not checked.

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Multiple CryptoDriverObjects to support multiple workspaces per service
Crypto_KeyElementSet supports an adapted variant of the ‘SHE Key Update Protocol’ [4]

Features Provided Beyond The AUTOSAR Standard
The AES CMAC Supports Round-key Reuse

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.3 Limitations

3.1.3.1 Cryptographic Algorithms and Modes

Only a subset of the stated algorithms and modes in the AUTOSAR SWS [1] are currently supported. The list of algorithms is described in Table 3-1.

3.1.3.2 Certificate Handling

Currently there is no implementation to parse and verify certificates. However the API is still available for compatibility reasons.

3.1.3.3 Key Generation

Currently there is no implementation to generate keys. However the API is still available for compatibility reasons.

3.1.3.4 AEAD Det Checks

The Det checks for AEAD differ from the table in [1]. There is a different handling required for the AEAD implementation. The parameter check is described in Table 3-8.

3.2 Initialization

Before any other functionality of the CRYPTO module can be called the initialization function `Crypto_30_LibCv_Init()` has to be called by the BSWM.

For manual null initialization of RAM variables the CRYPTO offers the function `Crypto_30_LibCv_InitMemory()` which can be called before the `Crypto_30_LibCv_Init()`.

3.3 Main Functions

The CRYPTO module implementation provides one main function. When the usage of asynchronous job processing is enabled, this main function has to be called cyclically on task level. The main function is responsible to start processing of the job.

3.4 SHE Key Update Protocol

The MICROSAR CRYPTO has the ability to update a key element by using a simplified SHE Key Update Protocol [4].

Note
In the actual implementation the key slot IDs encoded in the M1 message will not be evaluated, instead the key which is actually present in the referenced Crypto key is being used as authentication key.

3.4.1 Preconditions

When a key shall be updated using the SHE Key Update Protocol the

- WriteAccess has to be Encrypted
- ReadAccess has to be Denied
- Key material must be a 64 byte long concatenated M1|M2|M3 message
- If M4 and M5 are needed the “proof” key element has to be of size 48 bytes

3.4.2 Simplified SHE Key Update Protocol

The simplified protocol differs from the original protocol in the following:

- UID will be ignored
- CID will be ignored
- FID will be ignored

3.5 Implementation of esl_getBytesRNG

Currently the esl_getBytesRNG function is not provided within the Crypto_30_LibCv and the SecMod (Crypto Library) package.

Please implement this function within your project as specified in 5.3.1.

Expert Knowledge
By ensuring that initialization of the provided CRYPTO random number generator takes place before using elliptic curve operations, the esl_getBytesRNG implementation itself can call the configured Csm_RandomGenerate implementation.

3.6 Error Handling

3.6.1 Development Error Reporting

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

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

The reported CRYPTO ID is 114.

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

Service ID	Service
0x00	Crypto_30_LibCv_Init()
0x01	Crypto_30_LibCv_GetVersionInfo()
0x03	Crypto_30_LibCv_ProcessJob()
0x0E	Crypto_30_LibCv_CancelJob()
0x04	Crypto_30_LibCv_KeyElementSet()
0x05	Crypto_30_LibCv_KeyValidSet()
0x06	Crypto_30_LibCv_KeyElementGet()
0x0F	Crypto_30_LibCv_KeyElementCopy()
0x10	Crypto_30_LibCv_KeyCopy()
0x11	Crypto_30_LibCv_KeyElementIdsGet()
0x0D	Crypto_30_LibCv_RandomSeed()
0x07	Crypto_30_LibCv_KeyGenerate()
0x08	Crypto_30_LibCv_KeyDerive()
0x09	Crypto_30_LibCv_KeyExchangeCalcPubVal()
0x0A	Crypto_30_LibCv_KeyExchangeCalcSecret()
0x0B	Crypto_30_LibCv_CertificateParse()
0x12	Crypto_30_LibCv_CertificateVerify()
0x0C	Crypto_30_LibCv_MainFunction()

Table 3-4 Service IDs

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

Error Code	Description
0x00	CRYPTO_E_UNINIT
0x01	CRYPTO_E_INIT_FAILED
0x02	CRYPTO_E_PARAM_POINTER
0x04	CRYPTO_E_PARAM_HANDLE
0x05	CRYPTO_E_PARAM_VALUE

Table 3-5 Errors reported to DET

3.7 Key Derivation

This implementation supports the following Key Derivation Functions (KDF):

Key Derivation Functions
KDF_NIST_800-108 KDF in Counter Mode with SHA-256 as PRF
NIST.FIPS.186-4 Key Pair Generation Using Extra Random Bits with KDF in Counter Mode as RBG

Table 3-6 Key Derivation Functions

The KDF to use is specified by the algorithm key element (CRYPTO_KE_KEYDERIVATION_ALGORITHM) of the used parent key.
The following values can be configured:

ID	Name
1	CRYPTO_30_LIBCV_KDF_ALGO_KDF_SYM_NIST_800_108_CNT_MODE_SHA256
2	CRYPTO_30_LIBCV_KDF_ALGO_KDF_ASYM_NIST_FIPS_186_4_ERB

Table 3-7 Values for Key Derivation Algorithm Key Element

The behavior of the key derivation algorithm is configured by further specific key elements. These elements are
a password (CRYPTO_KE_KEYDERIVATION_PASSWORD),
a salt (CRYPTO_KE_KEYDERIVATION_SALT),
and the custom element label (CRYPTO_KE_CUSTOM_KEYDERIVATION_LABEL).

The number of iterations is automatically determined by the needed key length. The derived key is stored in the key element with id 1 of the given key.

3.8 AES CMAC Round-key Reuse

The AES CMAC supports the reuse of calculated AES round keys. To use this feature, a special key element, `CRYPTO_KE_CUSTOM_MAC_AES_ROUNDKEY` with id 0x81 needs to be configured for the used key with size of 256 byte. If so, the round keys will be stored in this key element after first calculation and will be used for following calculation jobs. Due to this, further calculations are faster than the first one.

For all keys without this element the round keys are calculated for every job.

Expert Knowledge
To save NVRAM, key elements of type `CRYPTO_KE_CUSTOM_MAC_AES_ROUNDKEY` should not be persisted. So, after every startup the round key is calculated during the first job.

3.9 Algorithm Parameter Overview

The required algorithm parameters are described in the following table. The required parameter differs from the table in [1].

Member / Service	inputPtr	inputLength	secondaryInputPtr	secondaryInputLength	tertiaryInputPtr	tertiaryInputLength	outputPtr	outputLengthPtr	secondaryOutputPtr	secondaryOutputLengthPtr	verifyPtr	output64Ptr	mode
HASH	U	U					F	F					SUF
MACGENERATE	U	U					F	F					SUF
MACVERIFY	U	U	F	F							F		SUF
ENCRYPT	U	U					UF	UF					SUF
DECRYPT	U	U					UF	UF					SUF
AEADENCRYPT	U	U	V	U~			UF	UF	F	F			SUF
AEADDECRYPT	U	U	V	U~	F	F	UF	UF			F		SUF
SIGNATUREGENERATE	UF	U F~					F	F					SUF
SIGNATUREVERIFY	UF	U F~	F	F							F		SUF
SECCOUNTERINCREMENT													
SECCOUNTERREAD												F	
RANDOMGENERATE							F	F					

S: member required in Start mode.
U: member required in Update mode.
V: member optional in Update mode.
F: member required in Finish mode.
~: no Det check required / 0 is a valid value.

Table 3-8 Overview of the required algorithm parameter

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
Crypto_30_LibCv.c	This is the main source file of the CRYPTO
Crypto_30_LibCv.h	This is the source file of the CRYPTO
Crypto_30_LibCv_Aead.c	This is source contains AEAD algorithms
Crypto_30_LibCv_Cipher.c	This is source contains cipher algorithms
Crypto_30_LibCv_Hash.c	This is source contains HASH algorithms
Crypto_30_LibCv_KeyManagement.c	This is source contains the CRYPTO´s key management functions
Crypto_30_LibCv_KeyManagement.h	This is header contains the CRYPTO´s key management functions
Crypto_30_LibCv_Mac.c	This is source contains MAC algorithms
Crypto_30_LibCv_Random.c	This is source contains PRNG algorithms
Crypto_30_LibCv_Services.h	This header file is used for the internal dispatcher
Crypto_30_LibCv_Signature.c	This is source contains signature algorithms

Table 4-1 Static files

4.1.2 Dynamic Files

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

File Name	Description
Crypto_30_LibCv_Cfg.c	This is configuration source file.
Crypto_30_LibCv_Cfg.h	This is configuration header file.

Table 4-2 Generated files

4.2 Critical Sections

Crypto uses the following critical sections:

> CRYPTO_30_LIBCV_EXCLUSIVE_AREA_0

This critical section protects workspace locking resources and so this section should never be interrupted by any other API of the Crypto module.

> CRYPTO_30_LIBCV_EXCLUSIVE_AREA_1

This critical section protects 32 Bit accesses and is only necessary in case of HW doesn't offer an atomic access.

> CRYPTO_30_LIBCV_EXCLUSIVE_AREA_2

This critical section protects the reading key access against a parallel key manipulation. The critical section should never be interrupted by any key manipulation method. So, most of the time, it is sufficient to prevent the current task to be interrupted by another task which could execute a Key Setting Method.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Services provided by CRYPTO

5.1.1 Crypto_30_LibCv_Init

Prototype	
`void Crypto_30_LibCv_Init (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Initializes the Crypto Driver.	
Particularities and Limitations	
Specification of module initialization > Interrupts are disabled. Module is uninitialized. This function initializes the module Crypto_30_LibCv. It initializes all variables and sets the module state to initialized.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-1 Crypto_30_LibCv_Init

5.1.2 Crypto_30_LibCv_InitMemory

Prototype	
`void Crypto_30_LibCv_InitMemory (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
The function initializes variables, which cannot be initialized with the startup code.	

Particularities and Limitations
Module is uninitialized. Initialize component variables at power up.
Call context
> TASK > This function is Synchronous > This function is Non-Reentrant

Table 5-2 Crypto_30_LibCv_InitMemory

5.1.3 Crypto_30_LibCv_GetVersionInfo

Prototype	
`void Crypto_30_LibCv_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information.	
Particularities and Limitations	
none Crypto_30_LibCv_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component.	
Call context	
> TASK|ISR > This function is Synchronous > This function is Reentrant	

Table 5-3 Crypto_30_LibCv_GetVersionInfo

5.1.4 Crypto_30_LibCv_ProcessJob

Prototype	
`Std_ReturnType Crypto_30_LibCv_ProcessJob (uint32 objectId, Crypto_JobType *job)`	
Parameter	
objectId [in]	Holds the identifier of the Crypto Driver Object.
job [in,out]	Pointer to the configuration of the job. Contains structures with job and primitive relevant information but also pointer to result buffers.

Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_NOT_VALID Request failed, the key is not valid.
	CRYPTO_E_QUEUE_FULL Request failed, the queue is full.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Process the received job.	
Particularities and Limitations	
none Performs the crypto primitive that is configured in the job parameter.	
Call context	
> TASK > This function is Reentrant	

Table 5-4 Crypto_30_LibCv_ProcessJob

5.1.5 Crypto_30_LibCv_CancelJob

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_CancelJob`** `(uint32 objectId, Crypto_JobType *job)`	
Parameter	
objectId [in]	Holds the identifier of the Crypto Driver Object.
job [in,out]	Pointer to the configuration of the job. Contains structures with user and primitive relevant information.
Return code	
Std_ReturnType	E_OK Request successful, job has been removed.
Std_ReturnType	E_NOT_OK Request failed, job could not be removed.
Functional Description	
Cancels the received job.	
Particularities and Limitations	
none This interface removes the provided job from the queue and cancels the processing of the job if possible.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-5 Crypto_30_LibCv_CancelJob

5.2 Key Management Functions

5.2.1 Crypto_30_LibCv_KeyCopy

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyCopy`** `(uint32 cryptoKeyId, uint32 targetCryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be the source element.
targetCryptoKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_READ_FAIL Request failed, read access was denied.
	CRYPTO_E_KEY_WRITE_FAIL Request failed, write access was denied.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element sizes are not compatible.
Functional Description	
Copy the key.	
Particularities and Limitations	
none Copies a key with all its elements to another key in the same crypto driver.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-6 Crypto_30_LibCv_KeyCopy

5.2.2 Crypto_30_LibCv_KeyElementCopy

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyElementCopy`** `(uint32 cryptoKeyId, uint32 keyElementId, uint32 targetCryptoKeyId, uint32 targetKeyElementId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be the source element.
keyElementId [in]	Holds the identifier of the key element which shall be the source for the copy operation.

targetCryptoKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
targetKeyElementId [in]	Holds the identifier of the key element which shall be the destination for the copy operation.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_READ_FAIL Request failed, read access was denied.
	CRYPTO_E_KEY_WRITE_FAIL Request failed, write access was denied.
	CRYPTO_E_KEY_EXTRACT_DENIED Request failed, not allowed to extract key material.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element sizes are not compatible.
Functional Description	
Copy key element.	
Particularities and Limitations	
none Copies a key element to another key element in the same crypto driver.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-7 Crypto_30_LibCv_KeyElementCopy

5.2.3 Crypto_30_LibCv_KeyElementIdsGet

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyElementIdsGet`** `(uint32 cryptoKeyId, uint32 *keyElementIdsPtr, uint32 *keyElementIdsLengthPtr)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose available element ids shall be exported.
keyElementIdsLengthPtr [in]	Holds a pointer to the memory location in which the number of key element in the given key is stored. On calling this function, this parameter shall contain the size of the buffer provided by keyElementIdsPtr. When the request has finished, the actual number of key elements is stored.
keyElementIdsPtr [out]	Contains the pointer to the array where the ids of the key elements shall be stored.
Return code	
Std_ReturnType	E_OK Request successful.

	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Used to retrieve information which key elements are available in a given key.	
Particularities and Limitations	
none -	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-8 Crypto_30_LibCv_KeyElementIdsGet

5.2.4 Crypto_30_LibCv_KeyElementSet

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyElementSet`** `(uint32 cryptoKeyId, uint32 keyElementId, const uint8 *keyPtr, uint32 keyLength)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be set.
keyElementId [in]	Holds the identifier of the key element which shall be set.
keyPtr [in]	Holds the pointer to the key data which shall be set as key element.
keyLength [in]	Contains the length of the key element in bytes.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_WRITE_FAIL Request failed, write access was denied.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element size does not match size of provided data.
Functional Description	
Sets a key element.	
Particularities and Limitations	
none Sets the given key element bytes to the key identified by cryptoKeyId. .	
Call context	

> TASK > This function is Synchronous > This function is Reentrant

Table 5-9 Crypto_30_LibCv_KeyElementSet

5.2.5 Crypto_30_LibCv_KeyValidSet

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyValidSet`** `(uint32 cryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key elements shall be set to valid.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Sets the key to valid.	
Particularities and Limitations	
none Sets the key state of the key identified by cryptoKeyId to valid.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-10 Crypto_30_LibCv_KeyValidSet

5.2.6 Crypto_30_LibCv_KeyElementGet

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyElementGet`** `(uint32 cryptoKeyId, uint32 keyElementId, uint8 *resultPtr, uint32 *resultLengthPtr)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be set.
keyElementId [in]	Holds the identifier of the key element which shall be set.
resultPtr [in]	Holds the pointer to the key data which shall be set as key element.
resultLengthPtr [in]	Contains the length of the key element in bytes.
Return code	
Std_ReturnType	E_OK Request successful.

	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_READ_FAIL Request failed, read access was denied.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the provided buffer is too small to store the result.
Functional Description	
This interface shall be used to get a key element of the key identified by the cryptoKeyId and store the key element in the memory location pointed by the result pointer.	
Particularities and Limitations	
none -	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-11 Crypto_30_LibCv_KeyElementGet

5.2.7 Crypto_30_LibCv_RandomSeed

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_RandomSeed`** `(uint32 cryptoKeyId, const uint8 *entropyPtr, uint32 entropyLength)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key for which a new seed shall be generated.
entropyPtr [in]	Holds a pointer to the memory location which contains the data to feed the entropy.
entropyLength [in]	Contains the length of the entropy in bytes.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Initialize the seed.	
Particularities and Limitations	
none This function generates the internal seed state using the provided entropy source. Furthermore, this function can be used to update the seed state with new entropy.	

Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 5-12 Crypto_30_LibCv_RandomSeed

5.2.8 Crypto_30_LibCv_KeyGenerate

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyGenerate`** `(uint32 cryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which is to be updated with the generated value.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Generates a key.	
Particularities and Limitations	
none This function shall dispatch the key generate function to the configured crypto driver object.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-13 Crypto_30_LibCv_KeyGenerate

5.2.9 Crypto_30_LibCv_KeyDerive

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyDerive`** `(uint32 cryptoKeyId, uint32 targetCryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which is used for key derivation.
targetCryptoKeyId [in]	Holds the identifier of the key which is used to store the derived key.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.

	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Derives a key.	
Particularities and Limitations	
none Derives a new key by using the key elements in the given key identified by the cryptoKeyId. The given key must contain the key elements for the password (CRYPTO_KE_KEYDERIVATION_PASSWORD), salt (CRYPTO_KE_KEYDERIVATION_SALT), algorithm (CRYPTO_KE_KEYDERIVATION_ALGORITHM) and the costum element label(CRYPTO_KE_CUSTOM_KEYDERIVATION_LABEL). The number of iterations is automatically determined by the needed key length. The derived key is stored in the key element with id 1 of the given key identified by targetCryptoKeyId.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-14 Crypto_30_LibCv_KeyDerive

5.2.10 Crypto_30_LibCv_KeyExchangeCalcPubVal

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyExchangeCalcPubVal`** `(uint32 cryptoKeyId, uint8 *publicValuePtr, uint32 *publicValueLengthPtr)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which shall be used for the key exchange protocol.
publicValuePtr [out]	Contains the pointer to the data where the public value shall be stored.
publicValueLengthPtr [in,out]	Holds a pointer to the memory location in which the public value length information is stored. On calling this function, this parameter shall contain the size of the buffer provided by publicValuePtr. When the request has finished, the actual length of the returned value shall be stored.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Calculation of the public value.	
Particularities and Limitations	
none Calculates the public value for the key exchange and stores the public key in the memory location pointed by the public value pointer.	

Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 5-15 Crypto_30_LibCv_KeyExchangeCalcPubVal

5.2.11 Crypto_30_LibCv_KeyExchangeCalcSecret

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyExchangeCalcSecret`** `(uint32 cryptoKeyId, const uint8 *partnerPublicValuePtr, uint32 partnerPublicValueLength)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which shall be used for the key exchange protocol.
partnerPublicValuePtr [in]	Holds the pointer to the memory location which contains the partners public value.
partnerPublicValueLength [in]	Contains the length of the partners public value in bytes.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Calculation of the secret.	
Particularities and Limitations	
none Calculates the shared secret key for the key exchange with the key material of the key identified by the cryptoKeyId and the partner public key. The shared secret key is stored as a key element in the same key.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-16 Crypto_30_LibCv_KeyExchangeCalcSecret

5.2.12 Crypto_30_LibCv_CertificateParse

Prototype
`Std_ReturnType` **`Crypto_30_LibCv_CertificateParse`** `(uint32 cryptoKeyId)`

Parameter	
cryptoKeyId [in]	Holds the identifier of the key slot in which the certificate has been stored.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Parse stored certificate.	
Particularities and Limitations	
none Parses the certificate data stored in the key element CRYPTO_KE_CERT_DATA and fills the key elements CRYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_CERT_SIGNATURE	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-17 Crypto_30_LibCv_CertificateParse

5.2.13 Crypto_30_LibCv_CertificateVerify

Prototype
`Std_ReturnType` **`Crypto_30_LibCv_CertificateVerify`** `(uint32 cryptoKeyId, uint32 verifyCryptoKeyId, Crypto_VerifyResultType *verifyPtr)`

Parameter	
cryptoKeyId [in]	Holds the identifier of the key which shall be used to validate the certificate.
verifyCryptoKeyId [in]	Holds the identifier of the key containing the certificate, which shall be verified.
verifyPtr [out]	Holds a pointer to the memory location which will contain the result of the certificate verification.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Certificate verification.	
Particularities and Limitations	
none Verifies the certificate stored in the key referenced by verifyCryptoKeyId with the certificate stored in the key referenced by cryptoKeyId.	

Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 5-18 Crypto_30_LibCv_CertificateVerify

5.3 Services needed by CRYPTO

5.3.1 esl_getBytesRNG

Prototype	
`eslt_ErrorCode` **`esl_getBytesRNG`** `(actU16 target_length, actU8* target)`	
Parameter	
target_length [in]	Holds the length of random bytes needed
target [in/out]	Holds the pointer to the location where the random bytes shall be stored. This buffer has to be at least ‘target_length’
Return code	
eslt_ErrorCode	ESL_ERC_NO_ERROR Request successful.
	ESL_ERC_ERROR Request failed.
Functional Description	
The esl_getBytesRNG is needed for elliptic curve algorithms to get pseudo randomly generated bytes. Currently there is no way to get entropy in the SecMod library so this function has to be implemented manually by the customer. This function shall provide ‘target_length’ amount of random bytes and write the output to the buffer given by ‘target’ pointer.	
Particularities and Limitations	
none	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-19 esl_getBytesRNG

Caution
Please include “ESLib.h” for the correct type definitions.

5.4 Services used by CRYPTO

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

Component	API
DET	Det_ReportError

Table 5-20 Services used by the CRYPTO

6 Configuration

In the CRYPTO the attributes can be configured according to/ with the following methods/ tools:

- Configuration in DaVinci Configurator 5 Pro

6.1 Configuration Variants

The CRYPTO supports the configuration variants

- `VARIANT-PRE-COMPILE`

The configuration classes of the CRYPTO parameters depend on the supported configuration variants. For their definitions please see the Crypto_30_LibCv_bswmd.arxml file.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
CSM	Crypto Service Manager
CRYIF	Crypto Interface
CRYPTO	Crypto Driver

Table 7-1 Glossary

7.2 Abbreviations

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

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com