MICROSAR Classic CRYPTO

Technical Reference

CRYPTO LIBCV

Version 14.00.00

Authors	vismss, vismwe, visenc, vismxe, viseag, viskju, coechsler
Status	Released

Document Information

History

Author	Date	Version	Remarks
vismss	2017-03-07	1.01.00	Initial creation of Technical Reference
vismwe	2017-08-31	2.01.00	Rework Document Updated supported features 2.1 Added chapter 2.1.3.3 Added chapter 2.6.2 Added chapter 2.7 Added chapter 2.13.2 Added description of all used EXCLUSIVE AREAS in chapter 3.2 Updated for behavior changes 4
vismwe	2017-09-27	3.00.00	Updated supported features 2.1 Added chapter 2.12.1 Added chapter 2.12.2 Updated chapter 2.4 for SHE Master Key support Updated chapter 3.1 for new file
vismwe	2017-10-27	3.01.00	Updated chapter 4 Added chapter 2.12.4 Added chapter 2.13.3 Added chapter 2.13.4 Grouped Key Usage 2.12 Grouped Algorithm 2.12.6
vismwe	2017-11-23	3.02.00	Updated supported features in 2.1 Added chapter 2.9 Updated chapter 2.13.3
vismwe	2017-12-19	3.03.00	Updated supported features and deviations 2.1 Update API Description for Key Derive Update chapter 2.7 Added chapter 2.8.1, 2.8.2 and 2.8.3
vismwe	2018-01-19	3.04.00	Updated supported features and deviations 2.1 Updated text for extended information's and descriptions 2.4 and 2.13.2 Updated chapter 2.12.1 and 7.2 Add Access Rights to 2.12.2 Added chapter 2.12.5, 2.13.11 and 3.3

Author	Date	Version	Remarks
vismwe	2018-02-13	3.05.00	Resolved ESCAN00097738 with adding chapter 2.16.1 Updated supported features and deviations 2.1 Added chapter 2.12.6, 2.12.7, 2.13.5, 2.13.6 and 2.16 Rework chapter 2.13.2 for new behavior Updated chapter 2.9, 2.12.1, 2.12.2 and 2.13.3
vismwe	2018-02-27	3.06.00	Updated supported features and deviations 2.1 Rework chapter 2.12.6, 2.13.5, 2.13.6 and 3.2
vismwe	2018-05-30	4.00.00	Updated supported features and deviations 2.1 Rework chapter 2.1.1 and 4 Add chapter 2.13.7
vismwe	2018-06-12	4.01.00	Updated supported features, deviations and limitations 2.1 Rework chapter 2.1.1, 2.4, 2.12.1, 2.12.2 and 2.12.6
vismwe	2018-08-22	4.02.00	Update architecture 1.1 Rework chapter 2.4, 2.16.1, 0 and 4
vismwe	2018-09-20	5.00.00	Updated supported features 2.1 Rework chapter 2.7, 2.13.3
vismwe	2018-11-23	5.01.00	Updated supported features 2.1 Rework chapter 4 Add chapter 2.12.8
vismwe	2018-11-29	5.02.00	Updated supported features 2.1 Rework chapter 2.13.3, 2.6.1 and 2.9
vismwe	2019-01-21	5.03.00	Updated supported features 2.1
vismwe	2019-02-15	5.04.00	Updated supported features 2.1
vismwe	2019-03-11	5.05.00	Update Reference Documents Update 1.1, 2.5, 2.13.5 and 4.4.1.1 Resolved ESCAN00100891
vismwe	2019-04-01	5.06.00	Updated supported features 2.1 Update 3.1 and 4
vismwe	2019-07-10	5.06.01	Update 2.13.6 and 2.16.1
vismwe	2019-07-16	6.00.00	Update 2.4, 4 Add 2.4.3, 2.4.4 and 2.4.5 Rework 3

Author	Date	Version	Remarks
vismwe	2019-08-06	6.01.00	Update supported features 2.1 Update 1.1 Rework chapter 2.4, 2.16.1, 4
vismwe	2019-12-20	7.00.00	Update supported features 2.1 Rework chapter 2.6.2, 2.7, 3.1 and 2.16.1
vismwe	2020-03-09	7.02.00	Update supported features 2.1 Add Chapter 2.11, 2.13.1 and 4.4.1.5 Reworked 2.5, 2.13.5 and 2.13.6
visenc	2020-04-08	7.03.00	Update supported features 2.1 Add chapter 2.10 Update chapter 2.12.2, 2.12.4 and 2.12.6 Update chapter 3.1
visenc	2020-04-17	8.00.00	Updated supported features 2.1 Update chapter 7.2
visenc vismwe	2020-05-06	8.01.00	Updated supported features 2.1 Updated custom primitive defines 2.13.1 Updated chapter 2.13.4 and 2.16.1
visenc	2020-06-05	8.02.00	Updated supported features 2.1 Add chapter 2.8.5 Updated chapter 4.2.10 Updated chapter 7.2
visenc	2020-06-17	8.03.00	Updated supported features 2.1 Add chapter 2.7 Updated chapter 4.2.10 Updated chapter 2.13.6
visenc	2020-07-03	8.04.00	Updated supported features 2.1 Updated custom primitive defines 2.13.1 Added chapter 2.13.8 Updated chapter 7.2
visenc vismwe	2020-07-23	8.05.00	Updated chapter 2.1, 2.1.1, 2.11, 2.13.6 and 2.16.1 Added chapter 2.12.3, 2.13.6.2 and 5.2
visenc	2020-09-02	9.00.00	Updated chapter 2.1, 2.1.2, 2.4, 2.11, 2.13.1, 2.13.6 Added chapter 2.4.6, 2.4.7 and 2.1.3.5
visenc vismwe	2020-10-30	9.01.00	Updated chapter 2.1, 2.7 and 2.9 Added chapter 2.7, 2.8.7, 2.8.8, 2.9.1 and 2.15
visenc	2020-11-24	9.02.00	Updated chapter 2.1, 2.10, 2.11, 2.13.5, 2.13.6, 2.16.1, 4.2.7, 4.2.10, 4.2.11, 4.2.12 and 7.2
vismxe vismwe	2021-01-13	9.03.00	Updated chapter2.5, 2.6.1, 2.11 and 2.13.1 Rework chapter 2.8, 2.12.6 and 2.16.1

Author	Date	Version	Remarks
visenc	2021-05-19	10.00.00	Updated chapter 2.1 Added chapter 2.1.3.8, 2.17, 4.4.2.2 and 4.4.2.3
vismwe	2021-09-10	10.01.00	Updated chapter 2.1, 2.7, 2.9, 2.12.1, 2.13.1 and 2.15 Added chapter 2.9.2
visenc	2021-12-08	11.00.00	Updated chapter 2.1, 2.8, 4.2.10 Added chapter 2.8.6
visenc viseag	2022-02-02	11.01.00	Added chapter 2.1.3.9, 4.4.2.4, 4.4.2.5, 2.1.3.4, 2.1.3.10 Updated chapter 3.1, 2.12.6, 2.4.2, 5.2, 2.13.4, 7.2, 2.13.8, 2.13.9 Updated chapter 3.1
alefarth visenc viseag viskju	2022-05-18	12.00.00	Product name updated to MICROSAR Classic, Updated chapter 2.1, 2.7 and 2.13.1 Updated chapter 2.13.7, added chapter 3.3.2 Updated chapter 2.6.2, 4.2.5 and 4.2.7
viskju	2022-11-15	12.01.00	Updated chapter 2.1, 2.6.1, 2.9, 2.17, 4.1.4
viskju	2023-01-30	12.02.00	Updated chapter 2.1, 2.8
viskju	2023-03-20	12.03.00	Updated chapter 2.1, 2.8.6
viskju	2023-04-03	12.04.00	Updated chapter 2.8.3, 2.13.1, 4.2.10
viskju	2023-04-14	13.00.00	Added chapter 2.1.3.11, Updated chapter 2.1.2, 2.5, 2.16.1, 3.2
viskju vikrio	2023-06-07	13.01.00	Updated chapter 2.1, 2.2, 2.5, 2.8, 2.9, 3.2, 2.16.1, Added chapter 2.1.3.12, 2.13.10, 2.18.1
coechsler	2023-31-07	13.02.00	Updated chapter 2.1, 2.8, 4.2.10 Added chapter 2.8.9, 2.13.4.1
coechsler	2023-08-21	13.03.00	Updated chapter 2.1 Added chapter 2.13.10.1, 2.1.3.13
viskju	2023-10-04	13.04.00	Added information about p521r1 key generation (2.1, 2.10) Improved description of AES output length requirements (2.13.7) Updated Table 2-13 (2.6.2) Added EDDSA to Abbreviations (7.2)
coechsler	2023-10-23	13.05.00	Added chapter Cybersecurity (6) Updated Table 2-27, Table 2-33, Table 2-34, Table 7-2, caution box in 2.6.2 Merged IsKeyOnCurve to chapter 2.13.11 Added chapter 2.9.3 ECDH Added limitation to 4.2.13, 4.2.14

Author	Date	Version	Remarks
coechsler	2024-01-08	13.05.01	Updated TCR names Updated ECBD 2.9.2, 2.9
coechsler	2024-01-19	13.06.00	Added chapter 2.13.12 SLH-DSA Added chapter 2.13.13 SM4 Updated 2.1 and added SM3 Added chapter 2.13.14 SM2 Signature Added SM2 KGA to 2.10
coechsler	2024-03-15	14.00.00	Added chapter 2.13.15 AEAD AES GCM

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_CryptoDriver.pdf	4.3.0
[2]	AUTOSAR	AUTOSAR_SWS_CryptoDriver.pdf	4.3.1
[3]	AUTOSAR	AUTOSAR_SWS_CryptoDriver.pdf	4.4.0
[4]	AUTOSAR	AUTOSAR_SWS_CryptoDriver.pdf	R19-11
[5]	AUTOSAR	AUTOSAR_SWS_CryptoDriver.pdf	R20-11
[6]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.3.0
[7]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	4.3.0
[8]	HIS	2009-04-01 SHE Functional Specification v1.1 (rev439)	1.1
[9]	NIST	NIST Special Publication 800-108	2009-10
[10]	NIST	FIPS PUB 186-4	2013-07
[11]	NIST	NIST Special Publication 800-56A	2
[12]	ISO	ISO 15118-2	2014-04-01
[13]	NIST	FIPS186-2	2000-01
[14]	NIST	SP800-90A	1
[15]	Vector	TechnicalReference_vSecPrim.pdf	2.1.0
[16]	NIST	NIST Special Publication 800-56C	1
[17]	NIST	NIST Special Publication 800-132	1
[18]	NIST	NIST Special Publication 800-38C	1
[19]	NIST	T. Taubert, C. Wood: SPAKE2+, an Augmented PAKE 2020-03	Version 00
[20]	ANSI	ANSI X9.63-2001	November 2001
[21]	ISO	ISO 15118-20	2021-10-08
[22]	GM	GM/T 0003.5-2012 Public key cryptographic algorithm SM2 based on elliptic curves - Part 5: Parameter definition	2012

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

Contents

Illustrations

Tables

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module 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. [7])

* 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 2.1.

1.1 Architecture Overview

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

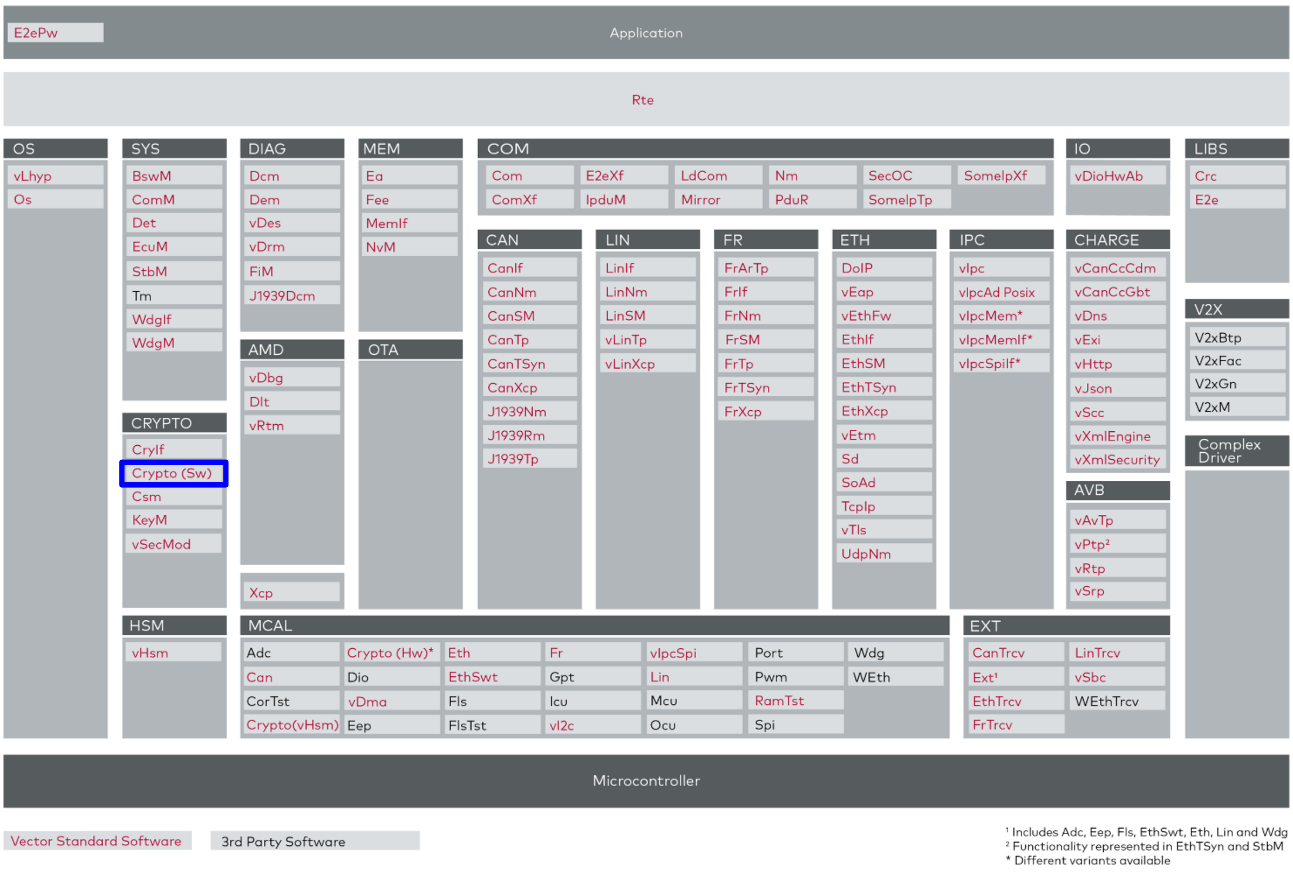

Figure 1-1 AUTOSAR 4.3 Architecture Overview

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

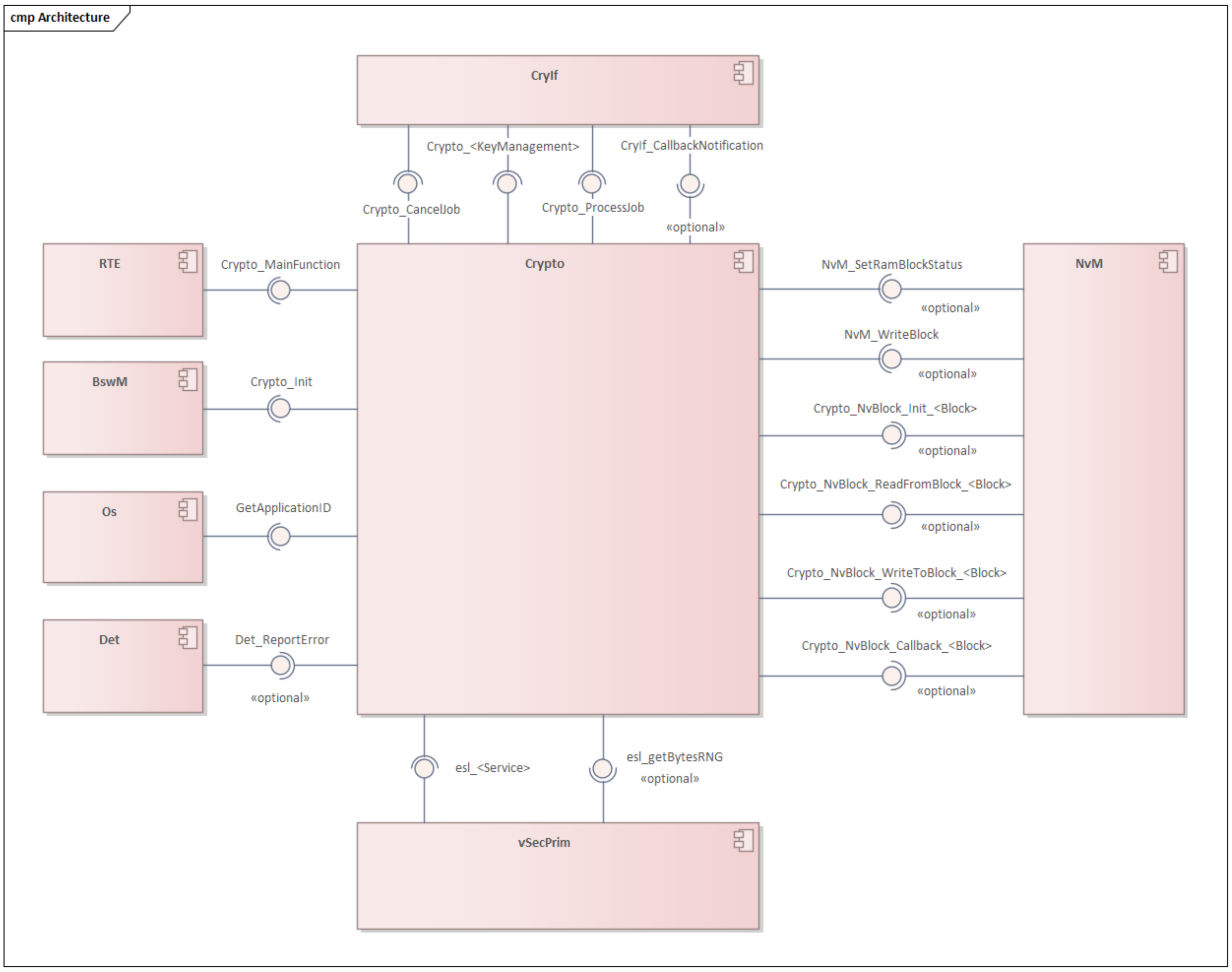

Figure 1-2 Interfaces to adjacent modules of the CRYPTO

2 Functional Description

2.1 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 2-1 Supported AUTOSAR standard conform features
- Table 2-2 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 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
AEAD: > AES-GCM > AES-CCM > ChaCha20-Poly1305
Cipher: > AES-128, AES-192 and AES-256 ECB/CBC/CTR with PKCS7 padding, ISO 9797-1 (OneWithZeros) padding and without padding > RSA PKCS1-V1.5; RSA PKCS1-V1.5 public key decryption > RSA OAEP and RSA CRT OAEP with SHA-1 and SHA2-256 pre-hashing (CRT is only available for decryption) > SM4 with block mode ECB/CBC
HASH: > SHA-1 > SHA2-256, SHA2-384, SHA2-512 > SHA3-224, SHA3-256, SHA3-384, SHA3-512 > SHAKE128, SHAKE256 > RIPEMD160 > MD5 > SM3
MAC: > CMAC AES-128, CMAC AES-256 > GMAC with AES-128, AES-192 and AES-256 > SipHash > HMAC-SHA-1 > HMAC-SHA2-256; HMAC-SHA2-384; HMAC-SHA2-512

Supported AUTOSAR Standard Conform Features
> HMAC-RIPEMD-160
PRNG: > FIPS 186-2 > DRBG-AES128 with DF and without DF > DRBG-AES256 with DF and without DF > DRBG-SHA2-512
Signature: > Ed25519 (pureEdDSA), Ed25519ctx (contextualized extension of the Ed25519 scheme) and Ed25519ph (hashEdDSA) > Ed448 (pureEdDSA), Ed448ph (hashEdDSA) > RSA PKCS1-V1.5, PKCS1-V1.5 CRT and PSS with SHA-1/SHA2-256 > NIST/ANSI/SECp256r1 and NIST/SECp384r1 without pre-hashing, SHA1, SHA2-256, SHA2-384 and SHA2-512 pre-hashing > SECp160r1 without pre-hashing > NIST/SECp521r1 with SHA2-512 > IsKeyOnCurve for SECp160r1, NIST/ANSI/SECp256r1, NIST/SECp384r1, NIST/SECp521r1 > IsKeyOnCurve for Ed25519 and Ed448 > SLH-DSA SHA2-128s > SM2 with Sm2p256v1-curve
Key Derive: > KDF in Counter Mode > KDF in Counter Mode with Appendix > Concatenation KDF (NIST 800-56A) > Certificate installation and update according to ISO15118-2 > Certificate installation according to ISO15118-20 > KDF X9.63 SHA2-256 and SHA2-512 > PBKDF2 HMAC SHA-1 and SHA2-256 > HKDF HASH Option 1 SHA-256 (one step variant) > HKDF HASH Option 1 SHA-512 (one step variant) > HKDF HMAC Option 2 SHA-256 (one step and two step variant) > HKDF HMAC Option 2 SHA-384 (one step and two step variant) > Spake2+ SECp256r1
Key Exchange: > ECDHE with ANSI Prime256v1, SECp256r1, SECp384r1, SECp521r1 > X25519 > X448 > Spake2+ > Elliptic Curve Burmester Desmedt with NIST/SECp244r1, NIST/ANSI/SECp256r1
Key Generate: > All symmetric keys > ECC with NIST/ANSI/SECp256r1, NIST/SECp384r1 and NIST/SECp521r1

Supported AUTOSAR Standard Conform Features
> Ed25519
> ECC with Sm2p256v1

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
Certificate handling
All algorithms not stated in Table 2-1
The access rights are mapped to different values according to 2.12.2
Usage of additional input data if redirection is used.
Virtual Key Elements
Crypto_JobType is according to [2]
Only Runtime Det CRYPTO_E_RE_ENTROPY_EXHAUSTED is supported

Table 2-2 Not supported AUTOSAR standard conform features

2.1.2 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' [8]
The AES CMAC Supports Round-key Reuse
Use vStdLib for memory access
Support deletion of key elements
KeyElementCopyPartial specified in [3]
Redirection specified in [3]
Processing of Key Jobs specified in [3]
Default Random Source specified in [4]
Write Once Keys
SHE Command Get ID
Long Term Workspace
Save and Restore Context specified in [5]
Multi-partition specified in [3]

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.3 Limitations

2.1.3.1 Cryptographic Algorithms and Modes

Only a subset of the stated algorithms and modes in the AUTOSAR SWS [1], [3] and [4] are currently supported. The list of algorithms is described in Table 2-1.

2.1.3.2 Certificate Handling

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

2.1.3.3 Parameter Det Checks

The Det checks for HASH, MAC and AEAD differ from the table in [1]. There is a different handling required for the implementation. The parameter check is described in 2.6.2.

2.1.3.4 AEAD AAD

The AEAD AES GCM, AES CCM and ChaCha20 Poly1305 encryption/decryption can optionally be called with additional authenticated data. If the AEAD AES GCM, AES CCM and ChaCha20 Poly1305 encryption/decryption is called with additional authenticated data, the data must be set during the first update call. Otherwise, the secondary input pointer must be initialized with a null pointer and secondary input length must be set to null.

2.1.3.5 Single Call Algorithms

The following algorithms are only support operations mode CRYPTO_OPERATIONMODE_SINGLECALL.

- SHE CMD_GET_ID
- AEAD AES CCM
- SLH_DSA

2.1.3.6 Key Derivation according to ISO15118-2

The ISO15118-2 certificate handling requires a validity check of the new ECC key pair specified in ISO15118-2 [V2G2-823]. This check is currently not part of the implementation. To be compliant to ISO15118-2 the validity of the ECC key pair needs to be checked by the caller. To check if the ECC key pair is valid, the new key pair could be used for generating and verifying a signature.

2.1.3.7 Key Length

Even if the length of the key is designed as uint32 in [1], the max key length is restricted to a value of 65535 (uint16 max value) due to internal limitations.

2.1.3.8 Save and Restore context

Save and restore context can only be configured per primitive for all driver objects and not per primitive and driver object.

2.1.3.9 Key Value and Validity Changed Callout

The callout for key value changed (See chapter 4.4.2.4) and key validity changed (See chapter 4.4.2.5) are not called when keys are set to their initial value. The keys are set to their initial values during start-up or initialization of the NvM blocks.

2.1.3.10 Verification results

Before starting the job, the user must set the verification result pointer to CRYPTO_E_VER_NOT_OK.

The verification is considered correct only if the return value of the job is equal to E_OK and the verification result pointer is equal to CRYPTO_E_VER_OK.

All other combinations can be interpreted as failed verification.

2.1.3.11 Multi-partition

The functionality to persist keys, as well as all SHE functionality is only available on the configured main partition. See also the hints at 2.4 Secure Hardware Extension, 2.5 Implementation of esl_getBytesRNG / Default Random Source, 2.12.4 Key Locking, 2.16.1 NvM Support and 3.2 Critical Sections.

AES CMAC Round-key Reuse (see 2.13.2) cannot be used when multi-partition functionality shall be used.

All partitions using the CRYPTO module have to use the same ASIL level.

2.1.3.12 RSA Encryption and Decryption

The RSA encryption and decryption does not allow multi update, since this is not specified for RSA.

2.2 Initialization

Before any other functionality of the CRYPTO module can be called, the initialization function `Crypto_30_LibCv_Init()` must be called by the BSWM on each partition where the module should be used.

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()`. See chapter Memory Init for Multi-Partition (2.18.1) for more details if multiple partitions are used.

2.3 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.

2.4 Secure Hardware Extension

The MICROSAR Classic CRYPTO has the ability to update a key element by using a simplified SHE [8]. The simplified protocol does not support all SHE features. The CRYPTO provides load key, load plain key, export RAM key, CMD_GET_ID and the CMD_DEBUG. The keys will be hold as plaintext in the CRYPTO. The NvM can be encrypted by user.

Caution
SHE functionality must only be used on the configured main partition.

2.4.1 Key Flags

- The Crypto supports the following key flags (FID). The initial value for the FID is 0. The key flags are hidden in the CRYPTO configuration of the SHE keys and cannot be configured. The keys can only be set by an update of the SHE key. The flags are included in the M2. For more information see [8].
  - WRITE PROTECTION
    If the flag is set to “1” the SHE key cannot be written anymore.
  - BOOT PROTECTION
    If the flag is set to “1” the key cannot be used if the Boot process is not finished.
  - DEBUGGER PROTECTION
    If the flag is set to “1” the key cannot be used if a Debugger is attached.
  - KEY USAGE
    If the flag is “1” the key can be used for CMAC otherwise for enc- and decryption.
  - WILDCARD
    If the flag is “1” the wildcard cannot be used within M1 for SHE key update.
  - CMAC USAGE
    This flag is only checked is KEY USAGE is “1” otherwise it shall be “0”, If the flag is set the key can only be used for CMAC verification. (Additional to the SHE standard)

The tables Table 2-4 and Table 2-5 show the usage of the key flags.

FID / SHE Key	WRITE PROTECTION	BOOT PROTECTION	DEBUGGER PROTECTION	KEY USAGE	WILDCARD	CMAC USAGE	Counter
SECRET_KEY		X*	X*				
MASTER_ECU_KEY	X	X	X		X		X
BOOT_MAC_KEY	X		X		X		X
BOOT_MAC	X		X		X		X
KEY_{n}	X	X	X	X	X	X	X
RAM_KEY_{Page}							
X* use flags from master key.							

Table 2-4 FID usage of SHE keys

For key usage, the counter is not checked. If the key is valid (e.g. set by the user), it can be used.

FID / Service	WRITE PROTECTION	BOOT PROTECTION	DEBUGGER PROTECTION	KEY USAGE	WILDCARD	CMAC USAGE	Counter
Decrypt		X	X	X			
Encrypt		X	X	X			
Mac Generate		X	X	X		X	
Mac Verify		X	X	X		X	
KeyElementSet	X	X	X		X		X
KeyElementGet		X*	X*				
*Read Access Rights are checked as well							

Table 2-5 Relevance of FIDs for the services

2.4.2 Load SHE Key

The SHE update implementation allows up to 255 SHE pages with up to 10 User and 1 RAM key per page. Secret key, Master key, Boot key as well as Boot Mac key are only allowed to exist on Page 0. This is illustrated in the following table.

Mapping SHE Key	SHE Id	Page
SECRET_KEY	0	0
MASTER_ECU_KEY	1	0
BOOT_MAC_KEY	2	0
BOOT_MAC	3	0
KEY_1_PAGE_M	4	M
KEY_2_PAGE_M	5	M
KEY_3_PAGE_M	6	M
KEY_4_PAGE_M	7	M
KEY_5_PAGE_M	8	M
KEY_6_PAGE_M	9	M
KEY_7_PAGE_M	10	M
KEY_8_PAGE_M	11	M
KEY_9_PAGE_M	12	M
KEY_10_PAGE_M	13	M
RAM_KEY_PAGE_M	14	M

Table 2-6 SHE Key Mapping

Before updating a SHE key, the corresponding update rights must be checked. The key rights depend on the used key and the used service. The dependencies are shown in Table 2-7 and Table 2-8.

Authentication Key Update Key	SECRET_KEY	MASTER_ECU_KEY	BOOT_MAC_KEY	BOOT_MAC	KEY_{n}	RAM_KEY_{Page}
SECRET_KEY						
MASTER_ECU_KEY		X				
BOOT_MAC_KEY		X	X			
BOOT_MAC		X	X			
KEY_{n}		X			X	
RAM_KEY_{Page}	X				X*	
X* only keys on the same SHE page						

Table 2-7 SHE Key Update Rights

SHE Key / Service	SECRET_KEY	MASTER_ECU_KEY	BOOT_MAC_KEY	BOOT_MAC	KEY_{n}	RAM_KEY_{Page}
Decrypt					X°	X
Encrypt					X°	X
Mac Generate					X°	X
Mac Verify			X		X°	X
KeyElementSet		X°	X°	X°	X°	X
Authentication Key	X	X	X		X	
KeyElementGet*	X*	X*	X*	X*	X*	X*^
CMD_DEBUG		X				
CMD_GET_ID		X				
*Read Access Rights allow reading / Key will be provided as plaintext key. ° Depends on the FID ^ Read Access Rights only allows encrypted reading / Key will be provided as encrypted key.						

Table 2-8 SHE Key Service usage

2.4.3 Load SHE Plain Key

In addition to other SHE keys, the RAM key can be either loaded in an encrypted manner as specified in 2.4.2 or as plaintext key. Therefore, the write access right needs to be WA_ALLOWED. If the access is set to WA_ALLOWED, the RAM key can still be loaded encrypted.

2.4.4 Export SHE RAM Key

The CRYPTO supports to export SHE RAM keys. A key can be exported via Crypto_30_LibCv_KeyElementGet. The service provides M1-M3 for a request with a 64 bytes buffer and M1-M5 for a 112Bytes buffer. The SHE RAM key can only be exported if it was set as plaintext. To export the key, the read access right needs to be RA_ENCRYPTED. The Secret Key is required for the export.

2.4.5 SHE DEBUG_CMD

The SHE Debug Command allows to set the SHE back to an initial state. The command deletes the current keys and restores the initial values. The command is executed on the SHE key and in the additionally configured debug command keys. This is only possible if none of these keys are write-protected.

- Get Challenge via KeyElementGet(SHE-Info-Key, CRYPTO_KE_CUSTOM_SHE_DEBUG_CMD)
  - This service checks if it is allowed to execute the DEBUG_CMD.

> Set Authorization via KeyElementSet(SHE-Info-Key, CRYPTO_KE_CUSTOM_SHE_DEBUG_CMD)
  - This service verifies the Authorization and set SHE to initial state.

2.4.6 SHE CMD_GET_ID

The SHE Get ID command returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

If MASTER_ECU_KEY is empty, the returned MAC is set to 0.

The primitive job is called as Mac generate job. There is a pre configuration for the SHE command get id job in the BSWMD.

> The challenge must be set in the input pointer with a length of 16 bytes

> The crypto key ID must be set to SHE master key and must also be valid

> The output pointer must have a length of 32 bytes

The output contains the following information:

Output = [15 bytes UID | 1 byte STATUS | 16 bytes CMAC]

2.4.7 SHE STATUS

The SHE status can be determined by SHE CMD_GET_ID.

Only the status of the boot protection and debugger protection is checked.

Bit	Name	Description
0	BUSY	Not supported
1	SECURE_BOOT	Not supported
2	BOOT_INIT	Not supported
3	BOOT_FINISHED	The bit is set when the secure booting has been finished by determining CRYPTO_KE_CUSTOM_SHE_BOOT_PROTECTION. See chapter 2.4.8
4	BOOT_OK	The bit is set when the secure booting has been finished by determining CRYPTO_KE_CUSTOM_SHE_BOOT_PROTECTION. See chapter 2.4.8
5	RND_INIT	Not supported
6	EXT_DEBUGGER	The bit is set if an external debugger is connected to the chip, this is determined by CRYPTO_KE_CUSTOM_SHE_DEBUGGER_PROTECTION.

		See chapter 2.4.8
7	INT_DEBUGGER	Not supported

Table 2-9 SHE Status Flags

2.4.8 Preconditions

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

SHE Key:

- Access Rights have to be
  - WA_ENCRYPTED for SHE key
  - WA_ALLOWED and RA_ENCRYPTED for SHE RAM key
- Key material must be a 64 byte long concatenated M1|M2|M3 message.
- The SHE key element needs to be configured with the size of 16 bytes.
- If M4 and M5 are needed, the "proof" key element must be of size 48 bytes. If the "proof" key element is not present, the SHE Key Update will be performed without calculating the proof.
- Enable check of FID.
  If the check is disabled, all flags will be interpreted as 0.
- Enable check of Counter.
  If the check is disabled, the counter will be ignored. If the counter is enabled the SHE key needs the "CRYPTO_KE_CUSTOM_SHE_COUNTER" key element, otherwise it can be excluded from configuration. The "CRYPTO_KE_CUSTOM_SHE_COUNTER" key element must be 4 bytes of size with an initialization value.

SHE Info Key:

The SHE info key holds the information for UID, Debugger- and Boot-Protection.

- UID
  If the UID is used the key element "CRYPTO_KE_CUSTOM_SHE_UID" must be configured with a size of at least 15 bytes.
- Debugger-Protection
  If the Debugger-Protection is used the key element "CRYPTO_KE_CUSTOM_SHE_DEBUGGER_PROTECTION" must be configured with the size of 1 byte. Otherwise it can be excluded from configuration. Setting this key element to 0x01u means that no debugger is attached and the keys with set Debugger-Protection will be accessible. Any other value will restrict the usage.
- Boot-Protection
  If the Boot -Protection is used the key element "CRYPTO_KE_CUSTOM_SHE_BOOT_PROTECTION" must be configured with the size of 1 byte. Otherwise it can be excluded from configuration. Setting this key element to 0x01u means that the boot process is interpreted as passed and the keys with Boot-Protection will be accessible. Any other value will restrict the usage.

Caution
The keys with Debugger- and Boot-Protection will be hold in RAM but will not be usable if the access protection is activated. The access protection is never set by the CRYPTO itself, so corresponding key elements need to be set by the integrator getting this information from the hardware.

SHE Keys cannot be copied or exported.

SHE Key are not validated automatically
SHE Keys with an initialization value are not set to valid during initialization like it is done for normal keys.

2.5 Implementation of esl_getBytesRNG / Default Random Source

The esl_getBytesRNG is required to provide random data for some algorithm provided by the vSecPrim [15]. The callout is required for RSA, ECC, Key Exchange and Key Generation. The exact list of the callout usage can be found in [15]. The callout esl_getBytesRNG can either be provided by the Default Random Source or implemented within your project. The Default Random Source can be defined by `CryptoDefaultRandomPrimitiveRef` and `CryptoDefaultRandomKeyRef`. For the Default Random Source only the DRBG can be used.

The callout is specified in 4.4.1.5.

Expert Knowledge
Ensure that the provided CRYPTO random number generator is initialized before RSA, ECC and Key Generation is used. This can be done by calling CSM_RandomSeed on `CryptoDefaultRandomKeyRef`.

Expert Knowledge
If the system uses Default Random Source and random number generation at the same time, the runtime can increase due to protection of the internal state of the RNG. This internal state is saved to the key referenced by `CryptoDefaultRandomKeyRef`. When Default Random Source and random number generation try to use the same key at the same time, the second user will return `CRYPTO_E_BUSY`.

This can be prevented by using different keys for Default Random Source and random number generation. If key locking is disabled, this is enforced.

Note
If the Det runtime error CRYPTO_E_RE_ENTROPY_EXHAUSTED is reported while random number generation with esl_getBytesRNG, the Default Random Source must be seeded. It must be ensured that the seed operation was successful.

If the Det runtime error CRYPTO_E_RE_GET_BYTES_RNG_ERROR is reported while random number generation with esl_getBytesRNG, any error occurred.

Caution
If the CRYPTO is used by multiple partitions, the `CryptoDefaultRandomKeyRef` must be different for each partition.

Default random keys can include key elements with persistency. As only the main partition will persist keys, the persistency of default random keys used by the non-main partitions must be handled manually (See 2.16.1).

2.6 Error Handling

2.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [6], if development error reporting is enabled (i.e. pre-compile parameter `CRYPTO_30_LIBCV_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 4. 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()
0x13	Crypto_30_LibCv_KeyElementCopyPartial()
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()
0x80	Crypto_30_LibCv_NvBlock_ReadFromBlock
0x81	Crypto_30_LibCv_NvBlock_WriteToBlock
0x82	Crypto_30_LibCv_NvBlock_Init
0x83	Crypto_30_LibCv_NvBlock_Callback
0x84	esl_getBytesRNG

Table 2-10 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 2-11 Errors reported to DET

Runtime Error Code	Description
0x03	CRYPTO_E_RE_ENTROPY_EXHAUSTED
0x30	CRYPTO_E_RE_GET_BYTES_RNG_ERROR

Table 2-12 Runtime Errors reported to DET

2.6.2 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,C	U~,C					F,C	F,C					SUFC
MACGENERATE	U,C	U~,C					F,C	F,C					SUFC
MACVERIFY	U,C	U~,C	F	F			C	C			F		SUFC
ENCRYPT	U,C	U,C					U,F,C	U,F,C					SUFC
DECRYPT	U,C	U,C					U,F,C	U,F,C					SUFC
AEADENCRYPT	U,C	U~,C	V	U~			U,F,C	U,F,C	F	F			SUFC
AEADDECRYPT	U,C	U~,C	V	U~	F	F	U,F,C	U,F,C			F		SUFC
SIGNATUREGENERATE	U,G,C	U, F~,C					FC	F,C					SUFC
SIGNATUREVERIFY	U,G,C	U, F~,C	F	F			C	C			F		SUFC
SECCOUNTERINCREMENT													SUF
SECCOUNTERREAD												F	SUF
RANDOMGENERATE							F,C	F,C					SUFC
RANDOMSEED	F	F											SUF
KEYGENERATE													SUF
KEYDERIVE													SUF
KEYEXCHANGE CALCPUBVAL							F	F					SUF
KEYEXCHANGE CALCSECRET	F	F											SUF

CERTIFICATE PARSE													SUF
CERTIFICATE VERIFY					F	F			F	F			SUF
KEYSETVALID													SUF

S: member required in Start mode.
U: member required in Update mode.
V: member optional in Update mode.
F: member required in Finish mode.
C: member required for Context Save/Restore operation
G: member optional in Finish mode.
~: no Det check required / 0 is a valid value.

Table 2-13 Overview of the required algorithm parameter

Caution
For signature generation and verification there is a special handling for input data. If the signature uses a pre-hashing, the input data can only be processed in update mode. Input data provided in finish mode will be ignored if pre-hashing is used.
If no pre-hashing is used, the input data can only be processed in finish mode.

Note
Some services (e.g. Hash) allow processing data of zero length (inputLength = 0). In these cases the inputPtr must still be a valid pointer (non-null-pointer). See Table 2-13 for the exact services.

2.7 Elliptic Curves

The following table shows information for the supported elliptic curves. The curves which are listed in a common table entry use the same coefficients.

Curve Name	field prime byte length (p-bytes)	group order byte length (n-bytes)
SECp160r1	20 Byte	21 Byte (161 Bit)
SECp224r1 NIST P-224	28 Byte	28 Byte
SECp256r1 NIST P-256 ANSI Prime256v1	32 Byte	32 Byte
Sm2p256v1	32 Byte	32 Byte
SECp384r1 NIST P-384	48 Byte	48 Byte
SECp521r1 NIST P-521	66 Byte	66 Byte

Table 2-14 Elliptic Curve Parameter Length

2.8 Key Derivation

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

Key Derivation Functions
KDF_NIST_800-108 KDF in Counter Mode with SHA2-256 as PRF as specified in [9]
NIST FIPS 186-4 Key Pair Generation Using Extra Random Bits with KDF in Counter Mode as RBG as specified in [10]
KDF NIST 800-56A One Pass C1E1S with “Single Step KDF” with HASH SHA2-256 as specified in [11]
ISO 15118-2 certificate installation and update as specified in [12]
ISO 15118-20 certificate installation as specified in [21]
KDF X9.63 with SHA2-256 and SHA2-512 [20]
PBKDF2 HMAC SHA-1 and SHA2-256 [17]
HKDF HASH Option 1 SHA-256 (one step variant) [16]
HKDF HASH Option 1 SHA-512 (one step variant) [16]
HKDF HMAC Option 2 SHA-256 (one step and two step variant) [16]
HKDF HMAC Option 2 SHA-384 (one step and two step variant) [16]
Spake2+ SECp256r1 Preamble [19]
HKDF Expand HMAC Option 2 SHA-256 (expansion step of two step variant) [16]
HKDF Expand HMAC Option 2 SHA-384 (expansion step of two step variant) [16]

Table 2-15 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	Value
1	CRYPTO_30_LIBCV_KDF_ALGO_KDF_SYM_NIST_800_108_CNT_MODE_SHA256	1u
2	CRYPTO_30_LIBCV_KDF_ALGO_KDF_ASYM_NIST_FIPS_186_4_ERB	2u
3	CRYPTO_30_LIBCV_KDF_ALGO_KDF_NIST_800_56_A_ONE_PASS_C1E1S_SINGLE_STEP_KDF_SHA256	3u
4	CRYPTO_30_LIBCV_KDF_ALGO_KDF_ISO_15118_CERTIFICATE_HANDLING	4u
5	CRYPTO_30_LIBCV_KDF_ALGO_KDF_X963_SHA256	6u
6	CRYPTO_30_LIBCV_KDF_ALGO_KDF_X963_SHA512	7u
7	CRYPTO_30_LIBCV_KDF_ALGO_PBKDF2_HMAC_SHA1	9u
8	CRYPTO_30_LIBCV_KDF_ALGO_PBKDF2_HMAC_SHA256	10u
9	CRYPTO_30_LIBCV_KDF_ALGO_HKDF_HMAC_SHA256	11u
10	CRYPTO_30_LIBCV_KDF_ALGO_SPAKE2_PLUS_P256R1	12u
11	CRYPTO_30_LIBCV_KDF_ALGO_HKDF_OPTION1_SHA256	13u
12	CRYPTO_30_LIBCV_KDF_ALGO_HKDF_HMAC_SHA384	14u
13	CRYPTO_30_LIBCV_KDF_ALGO_HKDF_OPTION1_SHA512	15u
14	CRYPTO_30_LIBCV_KDF_ALGO_KDF_ISO_15118_20_CERTIFICATE_HANDLING	16u
15	CRYPTO_30_LIBCV_KDF_ALGO_HKDF_EXPAND_HMAC_SHA256	17u
16	CRYPTO_30_LIBCV_KDF_ALGO_HKDF_EXPAND_HMAC_SHA384	18u

Table 2-16 Values for Key Derivation Algorithm Key Element

Derivation Algorithm	Derived key length
NIST 800 108 CNT SHA256	Determined by input data, see chapter 2.8.1
NIST FIPS 186 4 ERB	Determined by input data, see chapter 2.8.1
NIST 800 56A OnePass C1E1S SHA256	Fix 32-bytes
ISO15118-2	Fix 32-bytes
ISO15118-20	Fix 66-bytes (for SECP-521R1) Fix 56-bytes (for Curve448)
X9.63 SHA-256 and SHA-512	Determined by length of the target key
PBKDF2 HMAC SHA-1 and SHA-256	Determined by length of the target key
HKDF HASH Option 1 SHA-256/SHA-512	Determined by length of the target key

HKDF HMAC Option 2 SHA-256/SHA-384	Determined by length of the target key
Spake2Plus P256R1	Fix size, see chapter 2.8.8
HKDF Expand HMAC Option 2 SHA-256/SHA-384	Determined by length of the target key

Table 2-17 Key Derivation Algorithm and derived key length

2.8.1 Symmetric and Asymmetric Key Derivation

The behavior of the key derivation algorithm CRYPTO_30_LIBCV_KDF_ALGO_KDF_SYM_NIST_800_108_CNT_MODE_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_KDF_ASYM_NIST_FIPS_186_4_ERB is configured by further specific key elements.

These elements are:

> Password (CRYPTO_KE_KEYDERIVATION_PASSWORD)
> Salt (CRYPTO_KE_KEYDERIVATION_SALT)
> Custom element label (CRYPTO_KE_CUSTOM_KEYDERIVATION_LABEL)

Salt for symmetric/ asymmetric keys:

> Salt for symmetric keys:
`4 byte Context | 2 byte target key length in bytes (big endian)`

> Salt for asymmetric keys:
`4 byte Context | 2 byte target key length in bytes (big endian) | ((32 byte curve specific prime) - 1) or (32 byte curve specific order) - 1)`

The number of iterations is automatically determined by the required key length. The derived key is stored in the key element with id 1 of the given key.
The key derivation algorithms are only available on 32 and 64Bit Platforms.

If not stated differently, the number of iterations the KDF performs is automatically determined by the required key length and not to be confused with the value of the key element CRYPTO_KE_KEYDERIVATION_ITERATIONS.

2.8.2 Concatenation KDF

The key derive API is used to perform the concatenation KDF (CRYPTO_30_LIBCV_KDF_ALGO_KDF_NIST_800_56_A_ONE_PASS_C1E1S_SINGLE_STEP_KDF_SHA256) described in NIST 800-56A.

2.8.2.1 Preconditions

Before calling the API, the following key elements have to be set and must be marked as valid:

> **CRYPTO_KE_KEYDERIVATION_PASSWORD**
  This key element contains the private key.

> **CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO**
  This key element consists of the “Other Information” required for the algorithm.

> **CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY**
  This key element contains the public key needed for ECDH.

2.8.3 Smart Charge Communication (SCC)

The certificate handling of ISO15118-2 and ISO15118-20 is supported.

2.8.3.1 ISO15118-2 Certificate Handling

The key derive API is used to perform the key installation/update (CRYPTO_30_LIBCV_KDF_ALGO_KDF_ISO_15118_CERTIFICATE_HANDLING) described in ISO15118-2.

2.8.3.1.1 Preconditions

Before calling the API, the following key elements have to be set and must be marked as valid:

> **CRYPTO_KE_CUSTOM_SCC_CONTRACT_PUBLIC_KEY**
  This key element is the new public part of the encrypted new private key and consists of the concatenation of the two point coordinates.
  pubKey(64 bytes) = X(32 bytes) | Y(32 bytes)

> **CRYPTO_KE_CUSTOM_SCC_IV_AND_ENCRYPTED_PRIVATE_KEY**
  This key element consists of the concatenation of the IV and the encrypted new private key.
  Password(48 bytes) = IV(16 bytes) | ENC_PRIV(32 bytes)

> **CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY**
  This key element contains the public key needed for ECDH.

2.8.3.1.2 Certificate Installation

A provisioning key must be present at the key identified by cryptoKeyId. This can be achieved by setting the key element CRYPTO_KE_KEYDERIVATION_PASSWORD of cryptoKeyId with a key pair with KeyElementSet(). The targetCryptoKeyId specifies the destination for the key which shall be installed.

2.8.3.1.3 Certificate Update

A contract key must be present at the key identified by cryptoKeyId. This can be achieved by doing the certificate installation. The targetCryptoKeyId and cryptoKeyId are the same during certificate update.

2.8.3.2 ISO15118-20 Certificate Handling

The key derive API is used to perform the contract key installation (`CRYPTO_30_LIBCV_KDF_ALGO_KDF_ISO_15118_20_CERTIFICATE_HANDLING`) described in ISO15118-20. The implementation supports both curves (SECP-521R1 and Curve448). The curve to use is chosen based on the length of the key element `CRYPTO_KE_CUSTOM_SCC_IV_AND_ENCRYPTED_PRIVATE_KEY`.

2.8.3.2.1 Preconditions

Before calling the API, the following key elements have to be set and must be marked as valid:

> **CRYPTO_KE_CUSTOM_SCC_CONTRACT_PUBLIC_KEY**
>
> This key element is the public counterpart of the encrypted contract private key.
>
> > For curve SECP-521R1, it consists of the concatenation of the two point coordinates:
> >
> > pubKey$_{\text{SECP-521R1}}$(132 bytes) = X(66 bytes) | Y(66 bytes)
>
> > For Curve448, only one coordinate is used:
> >
> > pubKey$_{\text{Curve448}}$(56 bytes)

> **CRYPTO_KE_CUSTOM_SCC_IV_AND_ENCRYPTED_PRIVATE_KEY**
>
> This key element consists of the concatenation of the IV, the encrypted contract private key, and the corresponding tag value.
>
> Password$_{\text{SECP-521R1}}$ (94 bytes) = IV(12 bytes) | ENC_PRIV(66 bytes) | TAG(16 bytes)
>
> Password$_{\text{Curve448}}$ (84 bytes) = IV(12 bytes) | ENC_PRIV(56 bytes) | TAG(16 bytes)

> **CRYPTO_KE_CUSTOM_SCC_CONTRACT_AAD**
>
> This key element contains the concatenation of the PCID and the SKI.
>
> AAD(variable size) = PCID | SKI

> **CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY**
>
> This key element contains the public key needed for ECDH/EDDH.

2.8.3.2.2 Certificate Installation

A provisioning private key must be present at the key identified by cryptoKeyId. This can be achieved by setting the key element CRYPTO_KE_KEYDERIVATION_PASSWORD of cryptoKeyId with KeyElementSet(). The targetCryptoKeyId specifies the destination for the contract private key which shall be installed. The contract private key will be installed in the key element CRYPTO_KE_KEYDERIVATION_PASSWORD of the target key.

2.8.4 Key Derivation Function X9.63

The behavior of the key derivation algorithm
CRYPTO_30_LIBCV_KDF_ALGO_KDF_X963_SHA256 and
CRYPTO_30_LIBCV_KDF_ALGO_KDF_X963_SHA512 is configured by further specific key elements.
These elements are

> **CRYPTO_KE_KEYDERIVATION_PASSWORD**
> This key element contains the secret key.

> **CRYPTO_KE_KEYDERIVATION_SALT**
> This key element contains the optional salt.
> The salt key element needs to be excluded from configuration if it is not used.

The derived key is stored in the key element with id 1 of the given key.

2.8.5 Password-Based Key Derivation Function 2

The behavior of the key derivation algorithm
CRYPTO_30_LIBCV_KDF_ALGO_PBKDF2_HMAC_SHA1 and
CRYPTO_30_LIBCV_KDF_ALGO_PBKDF2_HMAC_SHA256 is configured by further specific key elements.
These elements are

> **CRYPTO_KE_KEYDERIVATION_PASSWORD**
> This key element contains the private key.

> **CRYPTO_KE_KEYDERIVATION_SALT**
> This key element contains the salt.

> **CRYPTO_KE_KEYDERIVATION_ITERATIONS**
> This key element contains the iterations counter. The key element must always be set with 4 bytes.

The derived key is stored in the key element with id 1 of the given key.

2.8.6 HKDF HASH Option 1 Sha-256 / Sha-512

The behavior of the key derivation algorithms CRYPTO_30_LIBCV_KDF_ALGO_HKDF_OPTION1_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_HKDF_OPTION1_SHA512 is configured by further specific key elements.

These elements are:

> **CRYPTO_KE_KEYDERIVATION_PASSWORD**
> This key element contains the private key.

> **CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO**
> This key element contains the optional additional information.

Optional key elements must be excluded, if they are not used.

The derived key is stored in the key element with id 1 of the given key.

2.8.7 HKDF HMAC Option 2 Sha-256 / Sha-384

The behavior of the key derivation algorithms CRYPTO_30_LIBCV_KDF_ALGO_HKDF_HMAC_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_HKDF_HMAC_SHA384 is configured by further specific key elements.

These elements are:

> **CRYPTO_KE_KEYDERIVATION_PASSWORD**
> This key element contains the private key.

> **CRYPTO_KE_KEYDERIVATION_SALT**
> This key element contains the optional salt.

> **CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO**
> This key element contains the optional additional information.

> **CRYPTO_KE_KEYDERIVATION_ITERATIONS**
> This key element contains the number of steps: 1 (one step variant) or 2 (two step variant)

Optional key elements must be excluded, if they are not used.

The derived key is stored in the key element with id 1 of the given key.

2.8.8 Spake2+ Preamble

The Spake2+ Preamble (CRYPTO_30_LIBCV_KDF_ALGO_SPAKE2_PLUS_P256R1) derives w0, w1 and L from a previously derived secret e.g. with the PBKDF. The length of the values is determined by the used curve, see Chapter 2.7.

> **Source Key**
  > **CRYPTO_KE_KEYDERIVATION_PASSWORD**
    This key element contains the concatenation of w0s and w1s.
    password (2 · curve n-bytes) = w0s(curve n-bytes) | w1s(curve n-bytes)
> **Target Key**
  > **CRYPTO_KE_CUSTOM_W0**
    This key element contains the derived w0 (curve n-bytes).
  > **CRYPTO_KE_CUSTOM_W1**
    This key element contains the derived w1 (curve n-bytes).
  > **CRYPTO_KE_CUSTOM_L**
    This key element contains the derived L ((2 · curve p-bytes +1) bytes).

2.8.9 HKDF Expand HMAC Option 2 Sha-256 / Sha-384

The HKDF Expand applies the key expansion step according to HKDF HMAC Option 2 Sha-256 / Sha-384, see Chapter 2.8.7. The derivation algorithms CRYPTO_30_LIBCV_KDF_ALGO_HKDF_EXPAND_HMAC_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_HKDF_EXPAND_HMAC_SHA384 are configured by further specific key elements.
These elements are:

> **CRYPTO_KE_KEYDERIVATION_PASSWORD**
  This key element contains the pseudorandom key (usually, the output from the extract step).
> **CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO**
  This key element contains the optional additional information.

2.9 Key Exchange

The implementation supports the following Key Exchange Algorithms (KEA):

Key Exchange Algorithm
ECDHE with ANSI Prime256v1 (equal to NIST P-256)
ECDHE with SECp256r1 (equal to NIST P-256)
ECDHE with SECp384r1 (equal to NIST P-384)
ECDHE with SECp521r1 (equal to NIST P-521)
ECDHE with X25519
ECDHE with X448
Spake2+
Elliptic Curve Burmester Desmedt with NIST/SEC P224R1
Elliptic Curve Burmester Desmedt with NIST/SEC/ANSI P256R1

Table 2-18 Key Exchange Algorithm

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

ID	Name	Value
1	CRYPTO_30_LIBCV_KEY_EXCHANGE_X25519	0u
2	CRYPTO_30_LIBCV_KEY_EXCHANGE_ANSIP256R1	1u
3	CRYPTO_30_LIBCV_KEY_EXCHANGE_SECP256R1	2u
4	CRYPTO_30_LIBCV_KEY_EXCHANGE_NISTP224R1_BD	3u
5	CRYPTO_30_LIBCV_KEY_EXCHANGE_SECP384R1	4u
6	CRYPTO_30_LIBCV_KEY_EXCHANGE_SPAKE2_PLUS_CIPHERSUITE_8	5u
7	CRYPTO_30_LIBCV_KEY_EXCHANGE_SPAKE2_PLUS_CIPHERSUITE_8_1	6u
8	CRYPTO_30_LIBCV_KEY_EXCHANGE_SECP521R1	7u
9	CRYPTO_30_LIBCV_KEY_EXCHANGE_X448	8u
10	CRYPTO_30_LIBCV_KEY_EXCHANGE_P256R1_BD	9u

Table 2-19 Values for Key Exchange Algorithm Key Element

The public key for p224r1, p256r1, p384r1 and p521r1 is stored in a special format like described in Table 2-36.

2.9.1 Spake2+

The CRYPTO supports one cipher suite of the Spake2+.

Suite	G	Hash	KDF	Mac
8	SECp256r1	SHA2-256	HKDF HMAC SHA2-256	CMAC AES128

Table 2-20 Spake2+ cipher suites

Caution

The Spake2+ uses a long term workspace for the calculation, see Chapter 2.15 for more information.

The long-term workspace will be locked with the call of CalcPubVal and will be released after the second call to CalcSecret (Verification).

To free the long-term workspace if the verification phase is not used, call the verification phase with valid input lengths and ignore the result.

The long-term workspace will also be released if there are internal errors while execution e.g. if the partner public value is invalid, but not if there are execution errors like a DET.

Note

The Spake2+ provides two variants of each cipher suite e.g. 8 and 8_1. The normal one, as specified in [19], is 8 and the one with the mode 1 (8_1) switches the verification keys, but otherwise uses the same functions as specified in the cipher suite in [19]. See [15] for more information.

Cipher suite 8 = ESL_SPAKE2P_MODE_CIPHERSUITE_8_1

Cipher suite 8_1 = ESL_SPAKE2P_MODE_CIPHERSUITE_8_2

Key Element	Party A	Party B	Operation	Info
CRYPTO_KE_CUSTOM_W0	X	X	PubVal	This key element contains w0. (curve n-bytes)
CRYPTO_KE_CUSTOM_W1	X	E	PubVal	This key element contains w1. (curve n-bytes)
CRYPTO_KE_CUSTOM_L	O	X	PubVal	This key element contains L. ((2 · curve p-bytes+1) bytes)
CRYPTO_KE_CUSTOM_ADDITIONAL_INFO	O	O	Secret1	This key element contains additional information.
CRYPTO_KE_KEYEXCHANGE_SHAREDVALUE	X	X	Secret1	This key element contains the shared value. (hash bytes÷2)
CRYPTO_KE_CUSTOM_VERIFICATION	X	X	Secret1	This key element contains the own verification mac. (mac bytes)
CRYPTO_KE_CUSTOM_VERIFICATION_RESULT	X	X	Secret2	This key element contains the verification result. (1 byte)
X: Required O: Optional key element. The key element must be excluded, if it is not used. E: Must be Excluded				

Table 2-21 Spake2+ Key Element usage

The length of the values is determined by the used algorithm: e.g. curve see Chapter 2.7., SHA2-256 = 32Bytes and CMAC-AES-128 = 16Bytes.

The existence of the key element CRYPTO_KE_CUSTOM_W1 is used to determine if the instance is Party A or Party B.

The key element CRYPTO_KE_CUSTOM_ADDITIONAL_INFO can contain the additional information for Info A, Info B and AAD. If the key element is used all information are optional but at least the length must be set for each information.

```
Info =    [ A-Length (4Bytes) | A-Data (Length Bytes)
          | B-Length (4Bytes) | B-Data (Length Bytes)
          | AAD-Length (4Bytes) | AAD-Data (Length Bytes)]
```

The Spake2+ Sequence is shown in the following diagram. The key confirmation step is optional.

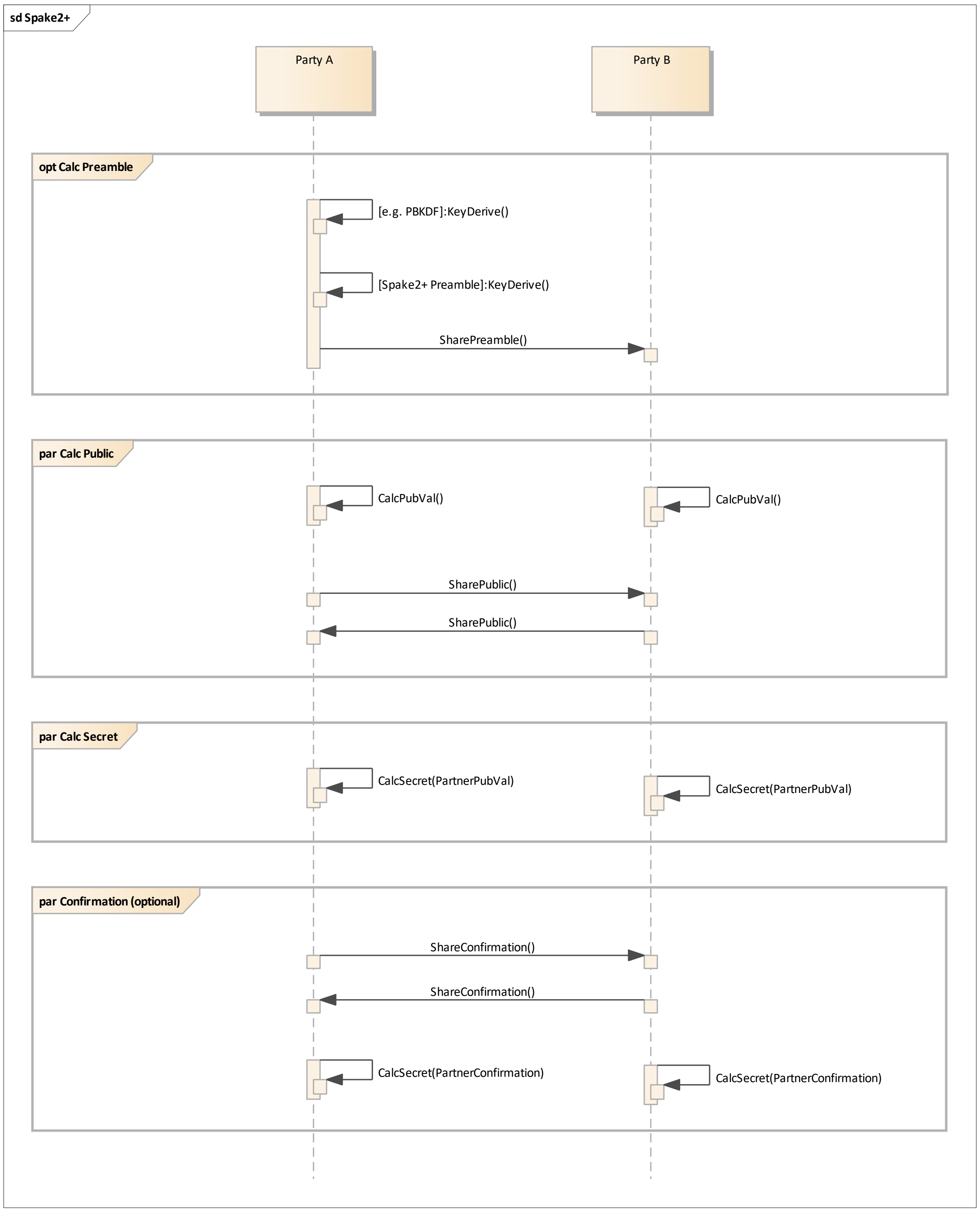

Figure 2-1 Spake2+ Control Sequence

2.9.2 Elliptic Curve Burmester Desmedt

The CRYPTO supports the Elliptic Curve Burmester Desmedt (ECBD) protocol. The ECBD protocol is a multi-party key agreement protocol which is implemented using Crypto_KeyExchangeCalcPubVal and Crypto_KeyExchangeCalcSecret.

Caution

The ECBD protocol uses a long-term workspace for the calculation, see Chapter 2.15 for more information.

The long-term workspace will be locked with the call to CalcPubVal and will be released after the last call to CalcSecret.

The long-term workspace will also be released if there are internal errors while execution e.g. if the partner public value is invalid, but not if there are execution errors like a DET.

To restart the protocol, the protocol must either be completed, failed or the public value calculation (after round 2) must be finished.

If the protocol shall be restarted after first round of public value calculation, call CalcSecret with valid parameters to unlock the workspace. Ignore the return value for this operation. After this, the protocol can be started again.

Note

The shared secret is stored into the key element CRYPTO_KE_KEYEXCHANGE_SHAREDVALUE. The format in which the shared secret is stored depends on the size of the key element as shown in Table 2-23.

Key Element	Operation	Info
CRYPTO_KE_CUSTOM_KEYEXCHANGE_NUM_ECU	PubVal Round 1	Total number of ECU's must be set. (1 byte)
CRYPTO_KE_CUSTOM_KEYEXCHANGE_ECU_ID	PubVal Round 1	ID of the own ECU must be set. (1 byte)
CRYPTO_KE_KEYEXCHANGE_OWNPUBKEY	PubVal Round 1	Own public value is set. ((2 · curve p-bytes) bytes)
CRYPTO_KE_KEYEXCHANGE_PRIVKEY	PubVal Round 1	Own private value is set. (curve n-bytes)
CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY	PubVal Round 2	Left partner public value must be set. ((2 · curve p-bytes) bytes)
CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY_2	PubVal Round 2	Right partner public value must be set. ((2 · curve p-bytes) bytes)
CRYPTO_KE_CUSTOM_KEYEXCHANGE_INTERMEDIATE	PubVal Round 2	Own intermediate value is set. ((2 · curve p-bytes) bytes)
CRYPTO_KE_KEYEXCHANGE_SHAREDVALUE	Secret Round 3	Shared value is set. (variable, see Table 2-23)

Table 2-22 ECBD Key Element usage

The protocol requires 3 or more nodes. The protocol consists of 3 rounds.

- Round 1: Each node
  - chooses a private key
  - computes a public key
  - broadcasts the public key to the left and right nodes
- Round 2: Each node
  - computes an intermediate key (second public key) depending on the received left and right partner public keys
  - broadcasts the intermediate key to all other nodes
- Round 3: Each node
  - computes the common secret depending on all intermediate values Hint: CalcSecret needs to be called for each incoming intermediate value.

The public values which are given to CalcSecret need to contain the partner's ECU ID and the partner's intermediate value.

```
publicValue =  [ EcuId (1 byte) |
                 intermediate value ((2 · curve p-bytes) bytes)]
```

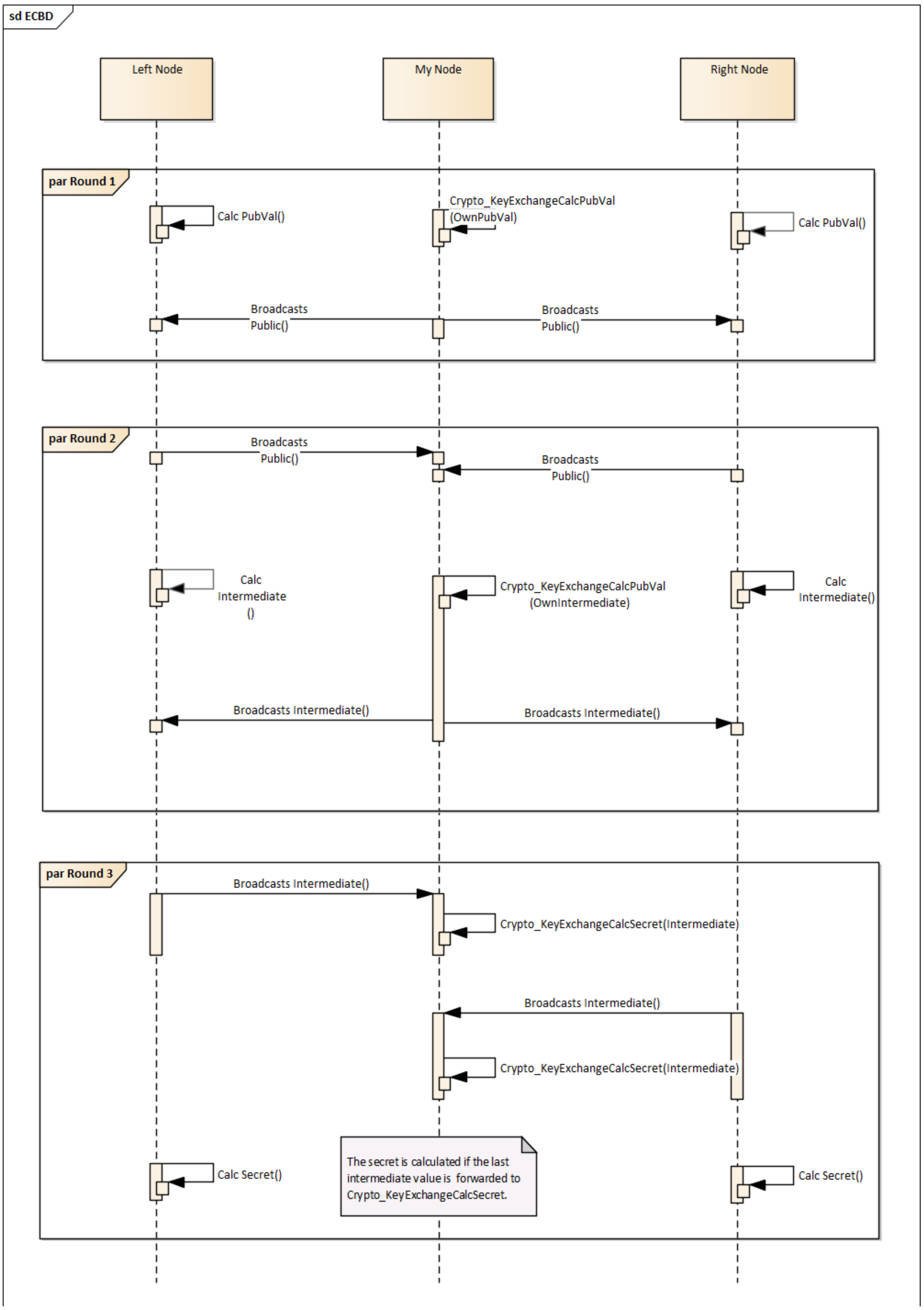
sd ECBD
Left Node
My Node
Right Node
par Round 1
Calc PubVal()
Crypto_KeyExchangeCalcPubVal (OwnPubVal)
Calc PubVal()
Broadcasts Public()
Broadcasts Public()
par Round 2
Broadcasts Public()
Broadcasts Public()
Calc Intermediate ()
Crypto_KeyExchangeCalcPubVal (OwnIntermediate)
Calc Intermediate()
Broadcasts Intermediate()
Broadcasts Intermediate()
par Round 3
Broadcasts Intermediate()
Crypto_KeyExchangeCalcSecret(Intermediate)
Broadcasts Intermediate()
Crypto_KeyExchangeCalcSecret(Intermediate)
Calc Secret()
The secret is calculated if the last intermediate value is forwarded to Crypto_KeyExchangeCalcSecret.
Calc Secret()

Figure 2-2 ECBD Control Sequence

2.9.3 Elliptic Curve Diffie-Hellman

The CRYPTO supports the Diffie-Hellman key exchange protocol with following elliptic curves:

- SECp256r1, ANSI Prime256v1
- SECp384r1
- SECp521r1

The shared secret is stored into the CRYPTO_KE_KEYEXCHANGE_SHAREDVALUE key element. The format in which the shared secret is stored depends on the size of the key element as shown in Table 2-23:

Shared Value Key Element size	Shared secret format	Partial Access necessary
> 2*size of private key	x-coordinate | y-coordinate	Yes
= 2*size of private key	x-coordinate | y-coordinate	No
< 2*size of private key and > size of private key	x-coordinate	Yes
= size of private key	x-coordinate	No

Table 2-23

2.10 Key Generate

The implementation supports the following Key Generate Algorithms (KGA):

Key Generate Algorithm
ECC with ANSI Prime256v1 (equal to NIST P-256 and SECp256r1)
ECC with SECp384r1 (equal to NIST P-384)
ECC with SECp521r1 (equal to NIST P-521)
Symmetric Keys
Ed25519
ECC with Sm2p256v1

Table 2-24 Key Generate Algorithm

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

ID	Name	Value
1	CRYPTO_30_LIBCV_KEY_GENERATE_SYMMETRIC	0u
2	CRYPTO_30_LIBCV_KEY_GENERATE_P256R1	1u
3	CRYPTO_30_LIBCV_KEY_GENERATE_P384R1	2u
4	CRYPTO_30_LIBCV_KEY_GENERATE_ED25519	3u
5	CRYPTO_30_LIBCV_KEY_GENERATE_P521R1	4u
6	CRYPTO_30_LIBCV_KEY_GENERATE_SM2P256V1	5u

Table 2-25 Values for Key Generate Algorithm Key Element

For symmetric key generation the target key length will be determined by the configured key element length. To generate keys with different sizes, it is required to configure different keys with the required key element size. The key will be generated in the key element with the id CRYPTO_KE_KEYGENERATE_KEY. The function esl_getBytesRNG is used to provide random data for the key generation service.

For elliptic curve key generation, the private key will be generated in the key element with the id CRYPTO_KE_KEYGENERATE_KEY and the public key in the element with the id CRYPTO_KE_KEYEXCHANGE_OWNPUBKEY. The public Key for p256r1, p384r1, p521r1 and sm2p256v1 is stored a special format like described in Table 2-36.

For Ed25519 key generation, the private key is generated in the key element with the identifier CRYPTO_KE_KEYGENERATE_KEY and the public key is generated in the element with the identifier CRYPTO_KE_KEYEXCHANGE_OWNPUBKEY. Both key elements must be 32 bytes long.

2.11 Random Seed

An algorithm for random number generation needs to be seeded before generation. This is required since the software algorithms are “Deterministic Random Number Generators” (DRNG). The algorithm for random number generation is defined by the algorithm key element (CRYPTO_KE_RANDOM_ALGORITHM) of the used parent key.
The following values can be configured:

ID	Name	Value
1	CRYPTO_30_LIBCV_RNG_FIPS_186_2_SHA1	0u
2	CRYPTO_30_LIBCV_RNG_NIST_800_90A_CTR_DRBG_AES128	1u
3	CRYPTO_30_LIBCV_RNG_NIST_800_90A_CTR_DRBG_AES128_DF	2u
4	CRYPTO_30_LIBCV_RNG_NIST_800_90A_HASH_DRBG_SHA_512	3u
5	CRYPTO_30_LIBCV_RNG_NIST_800_90A_CTR_DRBG_AES256	4u
6	CRYPTO_30_LIBCV_RNG_NIST_800_90A_CTR_DRBG_AES256_DF	5u

Table 2-26 Values for Key Random Algorithm Key Element

Note
If the Det runtime error CRYPTO_E_RE_ENTROPY_EXHAUSTED is reported while random number generation while RNG job processing, the RNG must be seeded. It must be ensured that the seed operation was successful.

For more information on Random Seed, refer to chapter 2.13.5 and 2.13.6.

2.12 Key Usage

To simplify the key usage this chapter describes some basic aspects.

2.12.1 Custom Key Elements

There are custom key elements defined, which are used to fulfill the algorithm requirements. These custom elements are defined in the Crypto_30_LibCv_Custom.h which need to be included in the Csm, e.g. via "`CsmCustomIncludeFiles`".

Custom Key Element	Value
CRYPTO_KE_CUSTOM_MAC_AES_ROUNDKEY	129u
CRYPTO_KE_CUSTOM_KEYDERIVATION_LABEL	130u
CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO	131u
CRYPTO_KE_CUSTOM_RSA_MODULUS	160u
CRYPTO_KE_CUSTOM_RSA_PUBLIC_EXPONENT	161u
CRYPTO_KE_CUSTOM_RSA_PRIVATE_EXPONENT	162u
CRYPTO_KE_CUSTOM_RSA_SALT	163u
CRYPTO_KE_CUSTOM_RSA_SALT_LENGTH	164u
CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY	3003u
CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY_2	3004u
CRYPTO_KE_CUSTOM_KEYEXCHANGE_INTERMEDIATE	3005u
CRYPTO_KE_CUSTOM_KEYEXCHANGE_NUM_ECU	3006u
CRYPTO_KE_CUSTOM_KEYEXCHANGE_ECU_ID	3007u
CRYPTO_KE_CUSTOM_SCC_CONTRACT_PUBLIC_KEY	3013u
CRYPTO_KE_CUSTOM_SCC_IV_AND_ENCRYPTED_PRIVATE_KEY	3014u
CRYPTO_KE_CUSTOM_RANDOM_PERSONALIZATION_STRING	3015u
CRYPTO_KE_CUSTOM_RANDOM_ADDITIONAL_INPUT	3016u
CRYPTO_KE_CUSTOM_RANDOM_NONCE	3017u
CRYPTO_KE_CUSTOM_RANDOM_RESEED_COUNTER	3018u
CRYPTO_KE_CUSTOM_SHE_COUNTER	3019u
CRYPTO_KE_CUSTOM_SHE_UID	3021u
CRYPTO_KE_CUSTOM_RSA_PRIME_P	3051u
CRYPTO_KE_CUSTOM_RSA_PRIME_Q	3052u
CRYPTO_KE_CUSTOM_RSA_EXPONENT_DP	3053u
CRYPTO_KE_CUSTOM_RSA_EXPONENT_DQ	3054u
CRYPTO_KE_CUSTOM_RSA_INVERSE_QI	3055u
CRYPTO_KE_CUSTOM_SHE_BOOT_PROTECTION	3056u
CRYPTO_KE_CUSTOM_SHE_DEBUGGER_PROTECTION	3057u
CRYPTO_KE_CUSTOM_LABEL	3058u
CRYPTO_KE_CUSTOM_SHE_DEBUG_CMD	3059u

CRYPTO_KE_CUSTOM_SCC_CONTRACT_AAD	3060u
CRYPTO_KE_CUSTOM_EDDSA_CONTEXT	3061u
CRYPTO_KE_CUSTOM_W0	3084u
CRYPTO_KE_CUSTOM_W1	3085u
CRYPTO_KE_CUSTOM_L	3086u
CRYPTO_KE_CUSTOM_SM2_ID	3092u
CRYPTO_KE_CUSTOM_SM2_GENERATE_PUBLIC_KEY	3093u

Table 2-27 Custom Key Elements

2.12.2 Key Element Access Rights

The CRYPTO driver provides the following key access rights with the shown security strength.

Access Right	Value	Access Strength	Description
ALLOWED	0	Lowermost	Accessible by plaintext.
ENCRYPTED	1		Encrypted external access only.
INTERNAL_COPY	2		No external access, only internal access.
DENIED	3	Highest	No external access, only internal job usage.

Table 2-28 Key Element Access Rights Ranking

Expert Knowledge
The read access right of key elements should be stricter or equal to the write access right to avoid secret keys easily read in plaintext.

The key element access rights will be checked according the following table.

Access	Key	Read Access Right	Write Access Right	Recommended Setting
KeyElementGet	Key	RA_ALLOWED	-	
	SHE RAM Key	RA_ENCRYPTED	-	
GET while ProcessJob	Key	ALL	-	
KeyElementSet	Key	-	WA_ALLOWED	
	SHE Key	ALL	WA_ENCRYPTED	RA_DENIED
	SHE RAM Key	ALL	WA_ALLOWED	RA_ENCRYPTED
KeyElementCopy* KeyCopy*	Source	RA right <= RA_INTERNAL_COPY	-	
	Destination	RA right >= Source Key	WA right <= WA_INTERNAL_COPY	

KeyDerive	Source	ALL	-	
	Destination	-	WA right <= WA_INTERNAL_COPY	
KeyExchange	Own Public	ALL	WA right <= WA_INTERNAL_COPY	RA_ALLOWED
	Private	ALL	WA right <= WA_INTERNAL_COPY	
	Shared	ALL	WA right <= WA_INTERNAL_COPY	
KeyGenerate	Key	ALL	WA right <= WA_INTERNAL_COPY	RA_ALLOWED
RandomSeed	Seed FIPS	-	WA_ALLOWED	
	Seed DRBG	RA_DENIED	WA_ALLOWED	
	ReseedCnt	ALL	WA_DENIED	

- : no effect, not used in function
ALL : key is accessed in function but all access rights are accepted
* : SHE keys cannot be copied

Table 2-29 Key Element Access Rights

2.12.3 Write Once Keys

The CRYPTO supports to configure key elements as write once. This allows that non-persist keys can only be written once after start-up and persist keys can only be written once for all time. Write once keys are not supported by all services the exact list can be found in the following table.

Service	Write Once Support
GET while ProcessJob	Not effected
SET while ProcessJob (Redirection)	Access will be denied
KeyElementGet	Not effected
KeyElementSet	Supported
KeyElementCopy	Supported
KeyElementCopyPartial	Supported
KeyCopy	Supported
KeyDerive	Supported
KeyExchange	Supported
KeyGenerate	Supported
RandomSeed	Access will be denied

Table 2-30 Write Once Support

2.12.4 Key Locking

The key is locked during the complete processing to hold the key and its key elements consistent. To allow a high performance it is possible to lock a key from several jobs for reading. A key can only be written if there is neither a read nor a write lock yet. If there is a write lock the key cannot be read.

Caution
If key locking is disabled, the user must ensure, that keys are only changed when they are not accessed (read or write) by any other user (different contexts or cores). To see which service functions modify keys, please see Table 2-31 Key Validity and the individual API definitions for the table entries.

If dedicated keys for each partition are used, the condition above only must be ensured for different contexts on the same core.

If a key is changed while being used, this can lead to wrong calculations of the using primitive.

Keys also must not be changed while they are read by NvM to persist them.

Caution
If key locking is disabled, immediate NvM Blocks (see 2.16.1) cannot be used.

Note
If default random source is used, the key will also be locked.

2.12.5 Pre-Configured Key Types

The CRYPTO provides a range of pre-configured key types. These pre-configured key types shall provide the most recommended usage scenarios. Further key types may be added by the user. Preconfigured key types could be used as template by duplicating them.

Not used key types will be skipped by the generator and have therefore no negative influence on the code and RAM size.

Expert Knowledge
To reduce the RAM usage configure the key type only with the required key elements. Also configure the key elements only with the required size.

2.12.6 Key Validity

The validity of the key is checked and modified by the service function according to the following table. If the key element has an init value the key element is set valid during the initialization excluding SHE keys.

Validity / Service	read key need to be valid	modified key is valid afterwards	modified key is invalid afterwards
ProcessJob Random Generate	X	X°	
ProcessJob All others	X		
ProcessJob Redirection	X		X
KeyElementGet	X		
KeyElementSet	X*		X
KeyElementCopy			X
KeyCopy			X
KeyValidSet (including ProcessJob)		X°	
KeyDerive (including ProcessJob)	X	X°	
KeyExchangeCalcSecret (including ProcessJob)	X		X
KeyExchangeCalcPubVal (including ProcessJob)	X		X
KeyGenerate (including ProcessJob)	X		X
RandomSeed (including ProcessJob)	X	X°	
esl_getBytesRNG	X	X°	
* SHE key update ° Callout for Key Validity is invoked			

Table 2-31 Key Validity

2.12.7 Key Element Delete

The Crypto supports a feature to delete a key element. To do this, the service Crypto_30_LibCv_KeyElementSet needs to be called with a valid keyPtr and the keyLength 0. The DET check for the keyLength will be disabled if the feature is enabled. If the key is set valid and read out the CRYPTO will return a key element with the length 0, which is not accepted by the most primitives.

Note
The key element will be overwritten once with zeros.

2.12.8 Redirection

The redirection allows to use key elements as in- and output-buffer. The streaming approach is allowed if only output keys are redirected. For the output redirection key elements can be written up to write access right WA_INTERNAL_COPY. For the input redirection key elements can be read up to RA_INTERNAL_COPY for the most services. If Encrypt, Decrypt, Aead-Encrypt or Aead-Decrypt is used only input keys with read access RA_ALLOWED can be used.

Property / Redirected Key	Need to be Valid	If invalid afterwards	Partial Access required
PRIMARY_INPUT	X		
SECONDARY_INPUT	X		
TERTIARY_INPUT	X		
PRIMARY_OUTPUT		X	X
SECONDARY_OUTPUT		X	X

Table 2-32 Redirection Key Properties

Note
Key Redirection is a feature of [3] therefore a crypto stack is required which also provides this feature including the change of the job structure.

The key element used for the primitive and the key elements used for redirection must be located in different crypto keys.

2.13 Algorithm

The algorithm specific handling is described in this chapter.

2.13.1 Custom Primitive Defines

To support primitive variants which are not defined by AUTOSAR, the custom defines listed in Table 2-33 and Table 2-34 are introduced and used within this Crypto driver.

The value of each define can be configured by changing the CryptoPrimitive-AlgorithmFamilyCustomId for the respective CryptoPrimitiveAlgorithmFamilyCustom or CryptoPrimitiveAlgorithmModeCustom container in the Cfg5. These containers are located under the CryptoPrimitives container and specified by [5]. While the containers themselves are preconfigured and cannot be changed, the IDs may be edited.

Custom Algorithm Mode Define	Custom Mode Container
CRYPTO_ALGOMODE_CUSTOM_CCM	CCM
CRYPTO_ALGOMODE_CUSTOM_MODE_1	Mode1
CRYPTO_ALGOMODE_CUSTOM_P160R1	P160r1
CRYPTO_ALGOMODE_CUSTOM_P224R1	P224r1
CRYPTO_ALGOMODE_CUSTOM_P256R1	P256r1
CRYPTO_ALGOMODE_CUSTOM_P384R1	P384r1
CRYPTO_ALGOMODE_CUSTOM_P521R1	P521r1
CRYPTO_ALGOMODE_CUSTOM_RSAES_OAEP_CRT	RSAES_OAEP_CRT
CRYPTO_ALGOMODE_CUSTOM_RSASSA_PKCS1_v1_5_CRT	RSASSA_PKCS1_v1_5_CRT
CRYPTO_ALGOMODE_CUSTOM_USE_DF	UseDF
CRYPTO_ALGOMODE_CUSTOM_ED25519_PURE	EdDsaEd25519Pure
CRYPTO_ALGOMODE_CUSTOM_ED25519_CTX	EdDsaEd25519Ctx
CRYPTO_ALGOMODE_CUSTOM_ED25519_PH	EdDsaEd25519Ph
CRYPTO_ALGOMODE_CUSTOM_ED448_PURE	EdDsaEd448Pure
CRYPTO_ALGOMODE_CUSTOM_ED448_PH	EdDsaEd448Ph
CRYPTO_ALGOMODE_CUSTOM_SHA2_128S	SHA2_128S
CRYPTO_ALGOMODE_CUSTOM_SM2P256V1	Sm2P256v1

Table 2-33 Custom Algorithm Modes

Custom Algorithm Family Define	Custom Family Container
CRYPTO_ALGOFAM_CUSTOM_CIPHER_SUITE_8	CipherSuite8

CRYPTO_ALGOFAM_CUSTOM_CMD_GET_ID	CmdGetId
CRYPTO_ALGOFAM_CUSTOM_DRBG	DRBG
CRYPTO_ALGOFAM_CUSTOM_ECCANSI	ECCANSI
CRYPTO_ALGOFAM_CUSTOM_ECCSEC	ECCSEC
CRYPTO_ALGOFAM_CUSTOM_FIPS186	FIPS186
CRYPTO_ALGOFAM_CUSTOM_HKDF	HKDF
CRYPTO_ALGOFAM_CUSTOM_ISO15118	ISO15118
CRYPTO_ALGOFAM_CUSTOM_ISO15118_20	ISO15118_20
CRYPTO_ALGOFAM_CUSTOM_MD5	MD5
CRYPTO_ALGOFAM_CUSTOM_POLY_1305	POLY1305
CRYPTO_ALGOFAM_CUSTOM_PADDING_PKCS7	PaddingPKCS7
CRYPTO_ALGOFAM_CUSTOM_SPAKE2_PLUS	SPAKE2PLUS
CRYPTO_ALGOFAM_CUSTOM_X25519	X25519
CRYPTO_ALGOFAM_CUSTOM_X448	X448
CRYPTO_ALGOFAM_CUSTOM_BD	BD
CRYPTO_ALGOFAM_CUSTOM_ED448	ED448
CRYPTO_ALGOFAM_CUSTOM_ISPUBLICKEYONCURVE	IsPublicKeyOnCurve
CRYPTO_ALGOFAM_CUSTOM_HKDF_EXPAND	HKDF_Expand
CRYPTO_ALGOFAM_CUSTOM_ISPRIVATEKEYONCURVE	IsPrivateKeyOnCurve
CRYPTO_ALGOFAM_CUSTOM_SLH_DSA	SLH_DSA
CRYPTO_ALGOFAM_CUSTOM_SM4	SM4

Table 2-34 Custom Algorithm Families

2.13.2 AES CMAC Round-key Reuse

The AES CMAC supports the reuse of calculated AES round keys. If this feature is enabled the round keys are hold in the driver object. The round keys are hold until a different job type is processed (not CMAC AES128) or the corresponding key has changed.

Expert Knowledge

To ensure the max speedup, configure a driver object for each AES CMAC key. The driver object can be used for several AES CMAC jobs with the same key.

2.13.3 RSA

The keys which are required by the RSA algorithm are stored in separate key elements which are defined in Table 2-27. The RSA PSS requires a salt for calculation which is always required. For the signature verification only, the salt length is required. If the salt length is not given or shall be calculated automatically the key element CRYPTO_KE_CUSTOM_RSA_SALT_LENGTH must be excluded from the used key.

Key Element / RSA Primitive	RSA_MODULUS	PUBLIC_EXPONENT	PRIVATE_EXPONENT	RSA_SALT	RSA_SALT_LENGTH	RSA_PRIME_P	RSA_PRIME_Q	RSA_EXPONENT_DP	RSA_EXPONENT_DQ	RSA_INVERSE_QI	LABEL
Signature Generate PKCS#1 V1.5	X		X								
Signature Verify PKCS#1 V1.5	X	X									
Signature Generate PKCS#1 V1.5 CRT						X	X	X	X	X	
Signature Generate PKCS#1 V2.2 PSS	X		X	X							
Signature Verify PKCS#1 V2.2 PSS	X	X			O*						
Encrypt PKCS#1 V1.5	X	X									
Decrypt PKCS#1 V1.5	X		X								
Encrypt PKCS#1 V2.2 OAEP	X	X									O*
Decrypt PKCS#1 V2.2 OAEP	X		X								O*
Decrypt PKCS#1 V2.2 OAEP CRT						X	X	X	X	X	O*
O*: Optional key element. The key element must be excluded, if it is not used.											

Table 2-35 RSA Key Element Usage

The RSA key elements need to be set as plain binary data without any format (e.g. ASN.1).

Expert Knowledge
The encryption and decryption does not allow multi update, since this is not specified for RSA.

2.13.4 ECDSA

The ECDSA requires a public key x and a public key y for verification. The public key x and public key y are stored in the signature key element. Since there is only one key element used, the key x and key y need to be stored concatenated. The size of the public key x and the public key y are the same so that the signature key element size is the double. The ECDSA requires hashed input data, if no pre hash is used the data must be hashed to be compliant with the algorithm.

Signature Key Element (Size 2L)	
public key x (Size L)	public key y (Size L)

Table 2-36 ECDSA Signature Key Element Storage

The signature for ECDSA is split in signature R and signature S. The signature is saved in the job output buffer. To do this, the signature R and signature S are concatenated. Signature R and signature S have the same size.

Signature Buffer	
Signature R	Signature S

Table 2-37 ECDSA Signature Storage

Note
The SECp160r1 signature contains an R and an S component, the length of the R component is always 160 bits and the length of the S component can be 161 bits, therefore the maximum size is fixed at 161 bits. The size of the output length is variable, if the first byte of the signature s is filled with zero padding, they can be deleted and the output length is set to 40 bytes (signature R and S). If the first byte of signature s is not filled with zero padding, the output length is set to 42 bytes (signature R and S). The signature verification service only requires that R and S must be equal in size. Since the service can process leading zeros on its own.

2.13.5 RNG FIPS

The random number generator is implemented according to [13].The FIPS 186-2 can only have one instance per driver object. The internal state of the RNG is stored in the driver object. The service RandomSeed is used to set the new entropy. If a new entropy is available, the RNG is seeded when the generation job is processed. During the random number generation, the key is locked for writing.

The length of the key element CRYPTO_KE_RANDOM_SEED_STATE needs to be set according to the used seed length. Since the FIPS 186-2 need to be reseeded after each startup the key element CRYPTO_KE_RANDOM_SEED_STATE shall not be persisted. The FIPS 186-2 requires an entropy length from at least 20 bytes, this is defined by [13].

Caution
If one key is used for several RNG jobs, only the first executed random instance gets seeded with this entropy. Therefore, it is highly recommended that each random instance has its own key, to avoid side effects. The RNG needs to be seeded after each startup.

Expert Knowledge
It is highly recommended to use the RNG DRBG instead of the RNG FIPS186.

2.13.6 RNG DRBG

The random number generator is implemented according to [14]. The chosen variants are DRBG-CTR AES 128 with DF, without DF, DRBG-CTR AES 256 with DF, without DF and DRBG-HASH SHA2-512.

The DRBG instance saves all required values in the key. Thereby it is possible to have multiple instances of the DRBG in one driver object.

There are two ways for the RNG seed state handling. The best way is to not persist CRYPTO_KE_CUSTOM_RANDOM_RESEED_COUNTER and CRYPTO_KE_RANDOM_SEED_STATE and seed the RNG after each startup. The other way is to persist CRYPTO_KE_CUSTOM_RANDOM_RESEED_COUNTER and CRYPTO_KE_RANDOM_SEED_STATE and continue random number generation with the old state. The DRBG gets instantiated with the first Crypto_30_LibCv_RandomSeed call.

Table 2-38 shows the required length of the key element CRYPTO_KE_RANDOM_SEED_STATE for the supported DRBG variants.

CRYPTO_KE_RANDOM_SEED_STATE	Length
CTR DRBG AES128 without DF	32 bytes
CTR DRBG AES128 with DF	32 bytes
CTR DRBG AES256 without DF	48 bytes
CTR DRBG AES256 with DF	48 bytes
HASH DRBG SHA 512	222 bytes

Table 2-38 Length for Key Random Seed State Element

Expert Knowledge
Use only one DRBG instance and reseed this instance as often as possible with new entropy to increase the numbers randomness. The entropy source is an important part to increase the numbers randomness.

Caution
If the key element CRYPTO_KE_RANDOM_SEED_STATE is configured to be persisted, the DRBG will continue random number generation based on the persisted state after restart. If the latest state is not written to NvM while shutdown, the DRBG will provide random numbers based on the last persisted state and the same random numbers are generated.

2.13.6.1 CTR DRBG

The CTR DRBG requires different input parameter for its operations. The required key elements are shown in the Table 2-39. The optional and recommended key elements must be excluded from the configured key if the input is not necessary. It is possible to configure DRBG-CTR AES 128/256 with DF and without DF at the same time. During the random number generation, the key is locked for writing.

DRBG Operation	Key Element	RANDOM_SEED_STATE	RANDOM_RESEED_COUNTER	RANDOM_ADDITIONAL_INPUT	RANDOM_PERSONALIZATION_STRING	RANDOM_NONCE
Without DF	Instantiation	X	X		R	
	Reseed	X	X	O		
	Generate Random Number	X	X	O		
With DF	Instantiation	X	X		R	X
	Reseed	X	X	O		
	Generate Random Number	X	X	O		
X: required R: Recommended O: Optional						

Table 2-39 CTR DRBG required key elements

For more information, e.g. about the length of the key elements in Table 2-39, please refer to [15], Chapter "Random Number Generation with DRBG".

2.13.6.2 HASH DRBG

The HASH DRBG requires different input parameter for its operations. The required key elements are shown in the Table 2-40. The optional and recommended key elements must be excluded from the configured key if the input is not necessary.

DRBG Operation	Key Element	RANDOM_SEED_STATE	RANDOM_RESEED_COUNTER	RANDOM_ADDITIONAL_INPUT	RANDOM_PERSONALIZATION_STRING	RANDOM_NONCE
HASH DRBG	Instantiation	X	X		R	X
	Reseed	X	X	O		
	Generate Random Number	X	X	O		
X: required R: Recommended O: Optional						

Table 2-40 HASH DRBG required key elements

For more information, e.g. about the length of the key elements in Table 2-40, please refer to [15], Chapter "Random Number Generation with DRBG".

2.13.7 AES Encryption/Decryption

The AES encryption/decryption supports key lengths of 128, 192 and 256 bits. If an AES job is executed the CRYPTO distinguishes the AES variant based on the given key length.

Enabling AES192 and AES 256 support
The support of AES192 and AES 256 needs to be activated in the general settings of the module configuration.

Note
Since the CRYPTO does not know the expected outputLength, the provided output buffer and its length need to be at least (AES block size * (N +1)), where N is the number of complete input blocks to be encrypted.
Depending on the padding mode, AES can add up to one complete block of encrypted bytes. Hence, the output buffer needs to provide for an additional block of (block size) many bytes.

This is also applicable for all AES based AEAD jobs (e.g. AES GCM).

2.13.8 ChaCha20 Poly1305

The ChaCha20 Poly1305 encryption/decryption supports only key length of 256 bit and nonce length of 96 bit. The nonce is stored in key element CRYPTO_KE_CIPHER_IV. In chapter 2.1.3.4 there is a limitation for the ChaCha20 Poly1305 with AAD.

Encryption:

The secondary input pointer contains the optional additional authenticated data
The output length must be greater than the input length.

Decryption:

The secondary input pointer contains the optional additional authenticated data
The output length must be greater than the input length.
The tertiary input length must be set to 16 bytes.

2.13.9 AEAD AES CCM

The AEAD AES CCM encryption/decryption supports only a single update step and a nonce length of 7 -13 byte. The nonce is stored in key element CRYPTO_KE_CIPHER_IV. The key length is limited to 128 bits. In chapter 2.1.3.4 there is a limitation for the AES CCM with AAD.

2.13.10 EDDSA

The EDDSA algorithms (ED25519 and ED448) support modes which can use a context. For this purpose, a special key element CRYPTO_KE_CUSTOM_EDDSA_CONTEXT is used. If it is set and marked as valid, the content will be treated as context for the algorithm. Otherwise, the context length will be set to zero.

Note
The length of the context must not exceed 255 bytes.

2.13.11 IsKeyOnCurve

The CRYPTO supports a IsKeyOnCurve functionality which verifies if a ECDSA or EDDSA key lies on a specific elliptic curve. The following elliptic curves are supported:

> SECp160r1

> SECp256r1, NIST P-256, ANSI Prime256v1

> SECp384r1, NIST P-384

> SECp521r1, NIST P-521

> ED25519

> ED448

The IsKeyOnCurve is provided via the SIGNATUREVERIFY service. The key to be verified needs to be provided as a key element CRYPTO_KE_SIGNATURE_KEY of a valid crypto key. In addition, the job parameters inputPtr and secondaryInputPtr need to be buffers with a valid address and a length greater than zero (dummy buffers). However, these buffers are not used for the computation but for compatibility of the SIGNATUREVERIFY service interface.

Note
Whether a private key lies on an elliptic curve cannot be checked per se, since private keys are not coordinates, but integers. However, private keys are checked for plausibility:

- ECDSA: based on the curve’s order
- EDDSA: based on the key length

2.13.12 SLH-DSA

The module supports Stateless Hash-Based Digital Signature Algorithm (SLH-DSA) with the following mode:

Mode	Key Length [bytes]	Signature Length [bytes]
SHA2-128s Verify	32	7 856

Table 2-41 SHL-DSA supported mode.

Caution
SLH-DSA is only supported in job operation mode CRYPTO_OPERATIONMODE_SINGLECALL.

2.13.13 SM4 Encryption/Decryption

The module supports the Chinese block cipher SM4. It's block length and cipher key length are both 16 bytes. The block modes ECB and CBC are supported. If ECB mode is used, the crypto key shall provide a cipher key KeyElement (e.g. CryptoKeyType CipherWithoutIv). For CBC, the crypto key shall provide a cipher key KeyElement and an initialization vector (IV) KeyElement (e.g. CryptoKeyType Cipher). Check the referenced KeyElement(s) for correct sizes of 16 bytes.

SM4 does not support padding. The overall input data size of each job must be a multiple of the block length (n*16 bytes) otherwise the job will fail in finish mode. However, when using streaming, the input data length may vary with each update step.

2.13.14 SM2 Signature

The module supports the Chinese signature algorithm SM2 with the elliptic curve Sm2p256v1 [22]. Similar to ECDSA (2.13.4), the public key is a concatenation of key x and key y. Further, the signature is a concatenation of signature R and signature S. SM2 uses various key elements for generation and verification, such as an optional key element for the user ID:

Signature Generate Key Elements	Key Element ID	Size [byte]
Private key	CRYPTO_KE_SIGNATURE_KEY	32
Public key	CRYPTO_KE_CUSTOM_SM2_GENERATE_PUBLIC_KEY	64
User ID (optional)	CRYPTO_KE_CUSTOM_SM2_ID	1 - 8191

Table 2-42 SM2 Signature Key Elements

Signature Verify Key Elements	Key Element ID	Size [byte]
Public key	CRYPTO_KE_SIGNATURE_KEY	64
User ID (optional)	CRYPTO_KE_CUSTOM_SM2_ID	1 - 8191

Table 2-43

2.13.15 AEAD AES GCM

The module supports AEAD AES GCM with an authentication tag length of 1-16 bytes.

Encrypt:

> If the authentication tag buffer (secondary Output) is 1-16 bytes, the tag will be written with the length of the given buffer.

> If the authentication tag buffer is larger than 16 bytes, the tag size will be set to 16 bytes.

Decrypt:

> If the authentication tag (tertiary Input) is 1-16 bytes, the given length will be used for the verification.

> If the authentication tag buffer is larger than 16 bytes, the job will fail.

2.14 Driver Objects and Jobs

The CRYPTO driver supports multiple driver objects. Each driver object has its own workspace. Algorithms share this workspace so that the size of the workspace depends on the configured algorithm with the highest RAM usage (e.g. RSA).

As soon as a job is started on a driver object for one algorithm, driver object is locked and therefore busy for another algorithm. If it is necessary to execute another job in between, a further driver object can be created and used for this job. As mentioned above, each driver object has its own workspace and can therefore be used to execute jobs parallel.

Unique Jobs
Jobs should only be owned and called by only one application since the driver cannot differ between one job from different callers. Ensure that a job is only called once at a time.

2.15 Long-Term Workspace

The CRYPTO supports long-term workspaces. These workspaces are required for algorithms which are executed over several service calls, e.g. the Spake2+ key exchange sequence. The long-term workspace will be locked at the start of the algorithm sequence and will be released at the end of the sequence. In some cases the workspace will be released if an error occurs.

Caution
The long-term workspace uses the cryptoKeyId for the locking mechanism. Each cryptoKeyId can only lock one long-term workspace.

If a service is restarted before the long-term workspace is released, the same workspace will be used.

Due to this behavior it must be prevented that there are requests for the same cryptoKeyId at the same time.

Note
The number of long-term workspaces can be determined by the use case. Useful information is:

> The number of keys which use long term workspaces
> The number of parallel requests

If all long-term workspaces are in usage the service will return with CRYPTO_E_BUSY.

Algorithm
Spake2+ Key Exchange
ECBD Key Exchange

Table 2-44 3.14 Long Term Workspace Algorithm List

2.16 Memory Access

2.16.1 NvM Support

The CRYPTO supports persisting of key elements to the NvM. Therefore, references to NvM blocks must be configured in CryptoNvStorage. The validation of the DaVinci Configurator 5 configures the block automatically to the needs of the CRYPTO. Key element persistence is triggered if a key is validated. This can be done by KeyValidSet (including ProcessJob), KeyDerive (including ProcessJob), RandomSeed (including ProcessJob) and ProcessJob Random Generate according to Table 2-31 Key Validity.

The CRYPTO provides an optional feature to notify the NvM that a referenced memory block has changed (e.g. by calling NvM_SetRamBlockStatus). This occurs if a key is set to valid which either can be triggered by the application or internally. The need for a valid key can be found in Table 2-31. If the feature is disabled, the CRYPTO module does not mark a block as modified (`NvM_SetRamBlockStatus`), it is up to the NvM to detect the need of writing the block.

The Crypto provides two modes for Block Processing:

> DEFERRED
  The block will only be marked as changed via `NvM_SetRamBlockStatus`.

> IMMEDIATE
  The block is marked as changed via `NvM_SetRamBlockStatus` and the `NvM_WriteBlock` is called. It is possible to overwrite the NvM write function and configure it for NvM writing of Crypto. For this purpose, the name of the function must be entered in the configuration `"/MICROSAR/Crypto_30_LibCv/Crypto/CryptoNvStorage/CryptoNvWriteBlockFctName"`. Therefore, there is a delay until the block is written to NvM.

Caution

Key element persistence is triggered if a key is validated (see Table 2-31 Key Validity).

Depending on the Block Processing mode the Crypto tries to trigger the write operation. If the request to start write operation fails (`NvM_SetRamBlockStatus` and `NvM_WriteBlock`), the Crypto service function will not return with an error. The Crypto will retry the operation in the next `Crypto_30_LibCv_MainFunction`.

If the NvM operation fails and one of the configured callbacks will report an error the write operation will not be retried. The failure needs to be detected by the customer using NvM. The retire of an NvM write error need to handle by the customer e.g. with NvM_WriteBlock for the effected block.

The Information of the NvM block status need to be retrieved via NvM.

Caution

If the multi-partition functionality is used, NvM operations on Crypto blocks must only be done on the main partition.

The Crypto itself will only call NvM APIs from the main partition. This means, that key elements are only persisted if they are validated (see Table 2-31 Key Validity) by the configured main partition (the partition where the NvM is located).

If such key elements are written by other partitions, they will not be persisted automatically. Also, a subsequent validation of the key on the main partition will not trigger the persist. Instead, the key can either manually be marked as changed (by calling `NvM_SetRamBlockStatus`) or directly written (by calling `NvM_WriteBlock`) .

▶ The Crypto provides two modes for Consistency Level:

> Level_None
  No checking for consistency (deprecated).
> Level_Detect
  The CRYPTO uses a CRC over the block key structure to detect changes. If changes are detected, the init values are restored. Otherwise the old block is used.

Expert Knowledge
For simplicity, persistent keys as well as non-persistent keys are processed and held in a variable called Crypto_30_KeyStorage. Therefore, the configured length of the corresponding NvM block can differ from the real length of this variable if there are keys which shall not be persisted. Persistent keys are located in the lowermost bytes of the Crypto_30_KeyStorage whereas non-persistent keys are located in the uppermost bytes.

Note
It is recommended to ensure that Crypto_30_LibCv data, which are stored in one or more NvM blocks configured by `/MICROSAR/Crypto_30_LibCv/Crypto/CryptoNvStorage/CryptoNvBlock`, are restored unchanged. The data integrity can be ensured e.g. by NvM using CRC.

The CRYPTO module does handle its configured NvM blocks. Therefore, the CRYPTO provides a callback for block initialization, reading from blocks and writing to blocks. These callbacks are specified in 4.4.1. All blocks need to be mapped to the NvM_ReadAll operation and it is recommended to map DEFERRED blocks to the NvM_WriteAll operation.

Caution
Persisted key elements must not be used until they are read from NvM (usually via Nvm_ReadAll). This is usually ensured by the initialization phases of the ECU.

Caution
Each Crypto_30_LibCv_NvBlock_Callback_<NvBlock> must always be called for a write operation concerning <NvBlock>, this includes NvM_WriteBlock and NvM_WriteAll. This can be either ensured by configuration (see 5.2) or by user code.

2.16.1.1 Layout Change Detection

The CRYPTO provides CRC calculation to detect Nv layout changes. The CRC Layout calculation includes:

> CRYPTO_ALIGN_KEY_STORAGE

- CRYPTO_NV_MANAGEMENT_DATA_PER_KEY
- For each Block
  - CRYPTO_NV_BLOCK_REVISION
  - For each Key within the Block
    - CRYPTO_KEY_NV_ID
    - For each Key Element within the Key
      - CRYPTO_KEY_ELEMENT_ID
      - CRYPTO_KEY_ELEMENT_SIZE

An explicit change for the Layout Detection can be accomplish by changing the CRYPTO_NV_BLOCK_REVISION.

Note
The CRYPTO_KEY_NV_ID is used to give a key an unique id for the complete lifecycle even if the CRYPTO_KEY_ID changed. The CRYPTO_KEY_NV_ID is used to identify a key within the Nv Block over several different configurations. Thereby it is possible to keep old NvBlock data instead of initializing them.

Caution
The required NvM size and key location depends on the configuration of persistent keys and its members. If any changes on persistent keys will be made or any persistent keys will be added or removed, the memory layout changes. This leads to the loss of the key material. Non-persistent keys do not affect the memory layout and therefore do not cause this change.

The loss of key material can be prevented by these two options:

- Configuration of new keys in a new NvM Block.
- Preallocation of additional keys for future use (e.g. SecOC keys for additional PDUs which are not known at the beginning). This is only applicable if the structure of the key is known before. This can be achieved by configuring these keys as empty keys from the beginning.

Deletion or change of keys is not possible. If this is required, old keys need to stay for compatibility reasons as configured and new keys have to be added as described above. It is highly recommend to set the old key to RA_DENIED and WA_DENIED.

2.16.2 vStdLib

The CRYPTO module can use the vStdLib to speed up extensive memory writings.

2.17 Save and Restore Context

The save context and restore context is a mode to interrupt the streaming mode after start or update to continue it again later. The workspace is provided in save mode and inserted again in restore mode. The customer has the option to output the workspace encrypted or in plain text, this is possible via the callouts 4.4.2.2 and 4.4.2.3. To export the data a valid buffer with a valid length is required, the buffer should be at least as long as the context. In restore mode the same length of the context is expected therefore the length of the exported data must be equal to the imported data. Redirection is ignored for save and restore context modes.

Note
This functionality can be activated by enabling
`/MICROSAR/Crypto_30_LibCv/Crypto/CryptoPrimitives/CryptoPrimitive/CryptoPrimitiveSupportContext`

Save Context:

A call to save the context is possible after the start or update mode, a failed safe call does not change the job status, therefore the job can continue without any restrictions. After a save call the job can be ended with Finish Job to start a new job, the finish call does not have to be successful, even a failed finish call is sufficient.

Note
The required buffer size is determined by the used algorithm workspace. For some algorithm only a subset of the workspace is saved and restored. To obtain the correct length of the separate workspace, a call with output length zero is expected. The return value is set to E_NOT_OK and the output length is overwritten with the required buffer length.

Note
A call to save the context is not possible if output redirection was used for the start or update mode.

Restore Context:

A restore context call is possible at any time as long as the driver object is not locked by another job. After resetting the workspace, it is possible to restart the job with the restore context mode and continue with the update mode. For Signature Generate primitive, the signature key must be re-set for the key to be valid. After a failed restore operation, the job must be restarted because the workspace may have been overwritten

Caution
The workspace context is provided as plaintext to the callouts 4.4.2.2 and 4.4.2.3. The workspace context contains cryptographic content e.g. keys, so the user should keep this data secure. This can be ensured e.g. by encryption of the content which can be implemented in the callouts.

Caution
The workspace context is exported and imported and must be protected for changes. The data integrity can be ensured e.g. by using CRC which can be implemented in the callouts.

2.18 Multi-Partition Support

The CRYPTO supports being used on multiple partitions, which can be located on different cores. The accessing partitions have to be configured properly in `/MICROSAR/Crypto_30_LibCv/Crypto/CryptoGeneral/CryptoEcucPartitionRef`.

Caution
The user must ensure, that a partition only calls the Crypto using driver objects which were assigned to the accessing partition during configuration.

A check for this can be done by enabling `/MICROSAR/Crypto_30_LibCv/Crypto/CryptoGeneral/CryptoMultiPartitionRuntimeChecks`

Caution
There is additional information in the feature specific chapters which must be considered:

- 2.4 Secure Hardware Extension
- 2.5 Implementation of esl_getBytesRNG / Default Random Source
- 2.12.4 Key Locking
- 2.16.1 NvM Support
- 3.2 Critical Sections

2.18.1 Memory Init for Multi-Partition

For each configured partition the CRYPTO generates one dedicated `Crypto_30_LibCv_InitMemory_<OsApplicationName>()` function which takes care of initializing the partition dependent init variables to 0.

Initializing can be done in the following ways:

- Startup core calls all available `Crypto_30_LibCv_InitMemory_<OsApplicationName>()` functions.
  - This requires that the startup core can write to the memory of all partitions.
  - This might require mapping the partition specific init variables to shared memory.
  - This would require a dedicated MPU regions for the init variables of the partitions.
- Each core calls the `Crypto_30_LibCv_InitMemory_<OsApplicationName>()` functions for its partitions.
  - This can be used when init variables shall be mapped to core local memory which might not be accessible from other cores.
- Do not call the `Crypto_30_LibCv_InitMemory_<OsApplicationName>()` at all when ZERO_INIT variables are already correctly initialized.
  - Initialization is usually done by vLinkGen or Compiler specific startup code.

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

3 Integration

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

3.1 Embedded Implementation

The delivery of the CRYPTO contains these source code files:

File Name	Description	Integration Tasks
Crypto_30_LibCv.c	This is the main source file of the CRYPTO	-
Crypto_30_LibCv.h	This is the header file of the CRYPTO	-
Crypto_30_LibCv_Aead.c	This source contains AEAD algorithms	-
Crypto_30_LibCv_AeadDecrypt.h	This header file is used for the internal AEAD dispatcher	-
Crypto_30_LibCv_AeadEncrypt.h	This header file is used for the internal AEAD dispatcher	-
Crypto_30_LibCv_Cipher.c	This source contains cipher algorithms	-
Crypto_30_LibCv_Decrypt.h	This header file is used for the internal cipher dispatcher	-
Crypto_30_LibCv_Encrypt.h	This header file is used for the internal cipher dispatcher	-
Crypto_30_LibCv_Hash.c	This source contains hash algorithms	-
Crypto_30_LibCv_Hash.h	This header file is used for the internal hash dispatcher	-
Crypto_30_LibCv_InternalApi	This header file is used for the internal key management functions	-
Crypto_30_LibCv_KeyDerive.c	This source contains key derive algorithms	-
Crypto_30_LibCv_KeyDerive.h	This header file is used for the internal key derive dispatcher	-
Crypto_30_LibCv_KeyExchange.c	This source contains key exchange algorithms	-
Crypto_30_LibCv_KeyExchange.h	This header file is used for the internal key exchange dispatcher	-
Crypto_30_LibCv_KeyGenerate.c	This source contains key generate algorithms	-
Crypto_30_LibCv_KeyGenerate.h	This header file is used for the internal key generate dispatcher	-

File Name	Description	Integration Tasks
Crypto_30_LibCv_KeyManagement.c	This source contains the CRYPTO´s key management functions	-
Crypto_30_LibCv_KeyManagement.h	This header contains the CRYPTO´s key management functions	-
Crypto_30_LibCv_KeySetValid.h	This header file is used for the internal key set valid dispatcher	-
Crypto_30_LibCv_Mac.c	This source contains MAC algorithms	-
Crypto_30_LibCv_MacGenerate.h	This header file is used for the internal mac dispatcher	-
Crypto_30_LibCv_MacVerify.h	This header file is used for the internal mac dispatcher	-
Crypto_30_LibCv_Random.c	This source contains PRNG algorithms	-
Crypto_30_LibCv_RandomGenerate.h	This header file is used for the internal PRNG dispatcher	-
Crypto_30_LibCv_RandomSeed.h	This header file is used for the internal PRNG seed dispatcher	-
Crypto_30_LibCv_Services.h	This header file is used for the internal functions	-
Crypto_30_LibCv_Signature.c	This source contains signature algorithms	-
Crypto_30_LibCv_SignatureGenerate.h	This header file is used for the internal signature dispatcher	-
Crypto_30_LibCv_SignatureVerify.h	This header file is used for the internal signature dispatcher	-
Crypto_30_LibCv_Custom.h	This header contains internal defines.	-
Crypto_30_LibCv_Curve.h	This header contains internal elliptic curve parameter declaration.	-
Crypto_30_LibCv_Curve.c	This source contains internal elliptic curve parameter definition.	-
Crypto_30_LibCv_Cfg.c	This is configuration source file.	-
Crypto_30_LibCv_Cfg.h	This is configuration header file.	-
Crypto_30_LibCv_MemMap.h	This header file contains the memory section mapping. In deliveries of MICROSAR Classic R27 and newer, this file is generated. In older releases, it is static.	-

Table 3-1 Implementation files

3.2 Critical Sections

Crypto uses the following critical sections:

> CRYPTO_30_LIBCV_EXCLUSIVE_AREA_0

▶ This critical section protects workspace locking resources, the job queue and the NvBlock state. Furthermore, it ensures the consistency of the global RAM variables for the last job and default random source data.

> CRYPTO_30_LIBCV_EXCLUSIVE_AREA_1

▶ This critical section protects key locking resources and ensures the consistency of the key data.

▶ If the CRYPTO is accessed by multiple cores, either key locking must be disabled (then this critical section implementation can be configured as `None`) or this critical section implementation has to be configured as Spinlock.

> CRYPTO_30_LIBCV_EXCLUSIVE_AREA_2

▶ This critical section protects long-term workspace locking resources.

▶ If the CRYPTO is accessed by multiple cores, either the services which use a long-term workspace (see 2.15) must only be used on one core or this critical section implementation has to be configured as Spinlock.

3.3 Best Practice

This chapter offers some best practice to simplify the crypto stack configuration.

3.3.1 Bottom to Top

The best way to configure keys, key types and key elements is to start configuration at the CRYPTO driver before doing this step in CryIf and Csm. Using this way, the DaVinci Configurator offers a solving action to propagate the keys from the CRYPTO driver to the upper layers.

This also applies to the CYPTO driver objects and CSM job assignment. First plan which CSM jobs shall be processed and check the calling conditions. Secondly create the CRYPTO driver objects with the CRYPO primitives to fulfill the conditions. Thirdly create the CSM Jobs and refence the CRYPTO driver objects.

3.3.2 Linker Options

The CRYPTO provides domain parameters for all supported elliptic curves, independent from whether the corresponding cryptographic primitives are enabled or not. In order to avoid increased ROM consumption, it is recommended to enable linker options, such that the linker will remove unused symbols, if this is not enabled by default.

3.4 Memory Mapping

The objects (e.g. variables, functions, constants) are assigned to a specific memory section.

The memory sections can be grouped according to the necessary access permissions. Within this document these groups are called Memory Section Groups.

For the CRYPTO the following groups can be identified:

- Memory Section Group **Constant**
- Memory Section Group **Shared**
- Memory Section Groups **PartitionLocal**

The following chapters explain the memory section groups of the CRYPTO.

3.4.1 Memory Section Group Constant

This shows the constant sections used by CRYPTO.

All parts of the CRYPTO need read access to these memory sections.

- CRYPTO_30_LIBCV_*_SEC_CODE
- CRYPTO_30_LIBCV_*_SEC_CONST_<size>

3.4.2 Memory Section Group Shared

This shows the variable memory sections which are shared between all partitions.

All parts of the CRYPTO need read and write access to these memory sections. If multiple partitions are accessing this section, it must not be cached.

- CRYPTO_30_LIBCV_*_SEC_VAR_NOINIT_<size>
- CRYPTO_30_LIBCV_*_SEC_VAR_ZERO_INIT_<size>

3.4.3 Memory Section Group PartitionLocal

This shows the partition local variable memory sections. There is one such group per configured partition.

Each partition needs read and write access to its PartitionLocal Memory Section Group. Other partitions don’t need read or write access to it.

- CRYPTO_30_LIBCV_*_SEC_<OS_APPLICATION_NAME>_VAR_*_<size>

Note
If only one partition is used the `<OS_APPLICATION_NAME>` is generated as `CRYPTO_SINGLE_PARTITION`.

4 API Description

For an interface overview please see Figure 1-2.

4.1 Services provided by CRYPTO

4.1.1 Crypto_30_LibCv_Init

Prototype	
`void Crypto_30_LibCv_Init (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Initializes the Crypto Driver. This function initializes the module Crypto_30_LibCv. It initializes all variables relevant for the partition and sets the module state for the partition to initialized. It has to be called once from the main partition and also from each other partition where the module is used.	
Particularities and Limitations	
Specification of module initialization Interrupts are disabled. Module is uninitialized.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-1 Crypto_30_LibCv_Init

4.1.2 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. Initialize component variables at power up. It should be called once from each partition where the module is used.	

Particularities and Limitations
Module is uninitialized.
Call context
> TASK > This function is Synchronous > This function is Non-Reentrant

Table 4-2 Crypto_30_LibCv_InitMemory

4.1.3 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. Crypto_30_LibCv_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component.	
Particularities and Limitations	
Configuration Variant(s): CRYPTO_30_LIBCV_VERSION_INFO_API == STD_ON	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-3 Crypto_30_LibCv_GetVersionInfo

4.1.4 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. If multi-partition support is enabled, only the objectId which belongs to the calling partition is allowed to be used.
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_KEY_SIZE_MISMATCH Request failed, a key element has the wrong size.
	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.
	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_EMPTY Request failed, uninitialized source key element.
	CRYPTO_E_ENTROPY_EXHAUSTION Request failed, the entropy is exhausted
	CRYPTO_E_JOB_CANCELED The service request failed because the synchronous Job has been canceled.
Functional Description	
Processes the received job. Performs the crypto primitive, that is configured in the job parameter.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for jobs with different job ids	

Table 4-4 Crypto_30_LibCv_ProcessJob

Caution
Any job must be called fully synchronously or asynchronously. It is not allowed to start the streaming mode synchronously and execute the finish step in asynchronous mode or vice versa. It is also not allowed to change other job parameters during streaming except the input and output buffer.

Caution
Any job must have been fully executed before starting a new job with the same job ID.

4.1.5 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. This interface removes the provided job from the queue and cancels the processing of the job if possible.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-5 Crypto_30_LibCv_CancelJob

4.2 Services for Key Management provided by CRYPTO

4.2.1 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. Copies a key with all its elements to another key in the same crypto driver. The key element can only be copied, if the destination key element write access right is less than WA_INTERNAL_COPY. Additional the read access right of the source must be less than RA_INTERNAL_COPY and the destination read access right must be higher or equal than the source read access right.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-6 Crypto_30_LibCv_KeyCopy

4.2.2 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_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. Copies a key element to another key element in the same crypto driver. The key element can only be copied, if the destination key element write access right is less than WA_INTERNAL_COPY. Additional the read access right of the source must be less than RA_INTERNAL_COPY and the destination read access right must be higher or equal than the source read access right.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-7 Crypto_30_LibCv_KeyElementCopy

4.2.3 Crypto_30_LibCv_KeyElementCopyPartial

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_KeyElementCopyPartial`** `(uint32 cryptoKeyId, uint32 keyElementId, uint32 keyElementSourceOffset, uint32 keyElementTargetOffset, uint32 keyElementCopyLength, 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.
keyElementSourceOffset [in]	Holds the offset of the of the source key element indicating the start index of the copy operation.
keyElementTargetOffset [in]	Holds the offset of the of the target key element indicating the start index of the copy operation.
keyElementCopyLength [in]	Holds the number of bytes that shall be copied.
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_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element sizes are not compatible.
	CRYPTO_E_KEY_EMPTY Request failed, uninitialized source key element.
Functional Description	
Copy key element partial. Copies a key element to another key element in the same crypto driver. The keyElementSourceOffset and keyElementCopyLength allows to copy just a part of the source key element into the destination. The offset of the target key is also specified with this function. The target key element needs to have partial access. The key element can only be copied, if the destination key element write access right is less than WA_INTERNAL_COPY. Additional the read access right of the source must be less than RA_INTERNAL_COPY and the destination read access right must be higher or equal than the source read access right.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-8 Crypto_30_LibCv_KeyElementCopyPartial

4.2.4 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.
keyElementIdsPtr [out]	Contains the pointer to the array where the ids of the key elements shall be stored.
keyElementIdsLengthPtr [out]	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.
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. The service provides a list of key element which are available in the given key.
Particularities and Limitations
-
Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 4-9 Crypto_30_LibCv_KeyElementIdsGet

4.2.5 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. Sets the given key element bytes to the key identified by cryptoKeyId. The key element can only be set, if the write access right is WA_ALLOWED or WA_ENCRYPTED. The access right need to be WA_ALLOWED for normal key, WA_ENCRYPTED for SHE key and WA_ALLOWED for SHE RAM key. If keyLength is zero the old value is deleted.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-10 Crypto_30_LibCv_KeyElementSet

4.2.6 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. Sets the key state of the key identified by cryptoKeyId to valid.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-11 Crypto_30_LibCv_KeyValidSet

4.2.7 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 [out]	Holds the pointer to the key data which shall be set as key element.
resultLengthPtr [in,out]	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_SMALL_BUFFER 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 to by the result pointer. The key element can only be provided, if the read access right is RA_ALLOWED. If the key element is configured to allow partial access, resultLengthPtr contains the amount of data which should be read from the key element. The behavior for partial access can be switched on or off in the configuration. If the read access right is RA_ENCRYPTED the SHE RAM key can be exported. (64 Bytes for M1-M3 and 112Bytes for M1-M5)	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-12 Crypto_30_LibCv_KeyElementGet

4.2.8 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.
Functional Description	
Initialize the seed. 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	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous	

> This function is Reentrant

Table 4-13 Crypto_30_LibCv_RandomSeed

4.2.9 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. This function shall dispatch the key generate function to the configured crypto driver object.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-14 Crypto_30_LibCv_KeyGenerate

4.2.10 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.	

Derives a new key by using the key elements in the given key identified by the cryptoKeyId. The key derivation function supports different algorithm which are specified by the algorithm key element(CRYPTO_KE_KEYDERIVATION_ALGORITHM).

- For CRYPTO_30_LIBCV_KDF_ALGO_KDF_SYM_NIST_800_108_CNT_MODE_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_KDF_ASYM_NIST_FIPS_186_4_ERB: 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 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 identified by targetCryptoKeyId. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY.
- For CRYPTO_30_LIBCV_KDF_ALGO_KDF_NIST_800_56_A_ONE_PASS_C1E1S_SINGLE_STEP_KDF_SHA256: The KDF is reads the algorithm configuration from the cryptoKeyId key elements. The KDF requires the following key elements the private key (CRYPTO_KE_KEYDERIVATION_PASSWORD), the public key (CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY) and the other Information (CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO). The derived key is set at the target key element (CRYPTO_KE_KEYDERIVATION_PASSWORD).
- For CRYPTO_30_LIBCV_KDF_ALGO_KDF_ISO_15118_CERTIFICATE_HANDLING: For certificate installation the cryptoKeyId need to be the root certificate. For certificate update the cryptoKeyId and targetCryptoKeyId need to be the same key Id. The KDF is reads the algorithm configuration from the cryptoKeyId key elements. The KDF requires the following key element the private key (CRYPTO_KE_KEYDERIVATION_PASSWORD). The following key elements need to be set at the targetCryptoKeyId: the contract/new public key (CRYPTO_KE_CUSTOM_SCC_CONTRACT_PUBLIC_KEY), the IV and encrypted private key (CRYPTO_KE_CUSTOM_SCC_IV_AND_ENCRYPTED_PRIVATE_KEY) and the public key (CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY). The derived key is set at the target key element (CRYPTO_KE_KEYDERIVATION_PASSWORD).
- For CRYPTO_30_LIBCV_KDF_ALGO_KDF_ISO_15118_20_CERTIFICATE_HANDLING:
  The KDF reads the algorithm configuration from the cryptoKeyId key elements. The following key element need to be set at the cryptoKeyId:
  the private key of the provision certificate (CRYPTO_KE_KEYDERIVATION_PASSWORD).
  The following key elements need to be set at the targetCryptoKeyId:
  the contract public key (CRYPTO_KE_CUSTOM_SCC_CONTRACT_PUBLIC_KEY),
  the IV, encrypted private key and tag (CRYPTO_KE_CUSTOM_SCC_IV_AND_ENCRYPTED_PRIVATE_KEY),
  the PCID and SKI (CRYPTO_KE_CUSTOM_SCC_CONTRACT_AAD) and the public key needed for ECDH (CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY).
  The derived key is set at the target key element (CRYPTO_KE_KEYDERIVATION_PASSWORD).
- For CRYPTO_30_LIBCV_KDF_ALGO_KDF_X963_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_KDF_X963_SHA512: Derives a key according to KDF X9.63 with the given CRYPTO_KE_KEYDERIVATION_PASSWORD(Secret) and optional CRYPTO_KE_KEYDERIVATION_SALT(Seed).
- For CRYPTO_30_LIBCV_KDF_ALGO_PBKDF2_HMAC_SHA1 and CRYPTO_30_LIBCV_KDF_ALGO_PBKDF2_HMAC_SHA256: 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 iteration element (CRYPTO_KE_KEYDERIVATION_ITERATIONS). The number of iterations is determined by the CRYPTO_KE_KEYDERIVATION_ITERATIONS element and must always be set with 4 bytes. The derived key is stored in the key element with id 1 of the given key identified by targetCryptoKeyId. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY.
- For CRYPTO_30_LIBCV_KDF_ALGO_HKDF_HMAC_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_HKDF_HMAC_SHA384:

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) iteration element (CRYPTO_KE_KEYDERIVATION_ITERATIONS) and optionally CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO (additional information). The iteration element specifies whether the one-step or two-step variant is to be used. The derived key is stored in the key element with id 1 of the given key identified by targetCryptoKeyId. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY. - For CRYPTO_KDF_ALGO_HKDF_EXPAND_HMAC_SHA256 and CRYPTO_KDF_ALGO_HKDF_EXPAND_HMAC_SHA384: The given key must contain the key elements for the password (CRYPTO_KE_KEYDERIVATION_PASSWORD) and optionally CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO (additional information). The derived key is stored in the key element with id 1 of the given key identified by targetCryptoKeyId. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY. - For CRYPTO_30_LIBCV_KDF_ALGO_HKDF_OPTION1_SHA256 and CRYPTO_30_LIBCV_KDF_ALGO_HKDF_OPTION1_SHA512: The given key must contain the key elements for the password (CRYPTO_KE_KEYDERIVATION_PASSWORD), algorithm (CRYPTO_KE_KEYDERIVATION_ALGORITHM) and optionally CRYPTO_KE_CUSTOM_KEYDERIVATION_ADDITIONAL_INFO (additional information). The Algorithm element specifies which hash variant is to be used. The derived key is stored in the key element with id 1 of the given key identified by targetCryptoKeyId. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY. - For CRYPTO_30_LIBCV_KDF_ALGO_SPAKE2_PLUS_P256R1: The given key must contain the key element for the password (CRYPTO_KE_KEYDERIVATION_PASSWORD). The derived key is stored in the key element with id 1 of the given key identified by targetCryptoKeyId. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY.
Particularities and Limitations
-
Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 4-15 Crypto_30_LibCv_KeyDerive

4.2.11 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. Calculates the public value for the key exchange and stores the public key in the memory location pointed to by the public value pointer. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-16 Crypto_30_LibCv_KeyExchangeCalcPubVal

4.2.12 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.
Functional Description	
Calculation of the secret.	

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. The write access right for the destination key element must be less than or equal to WA_INTERNAL_COPY.
Particularities and Limitations
-
Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 4-17 Crypto_30_LibCv_KeyExchangeCalcSecret

4.2.13 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. 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		
Particularities and Limitations		
Currently there is no implementation to parse certificates. However, the API is still available for compatibility reasons.		
Call context		
> TASK > This function is Synchronous > This function is Reentrant		

Table 4-18 Crypto_30_LibCv_CertificateParse

4.2.14 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. Verifies the certificate stored in the key referenced by verifyCryptoKeyId with the certificate stored in the key referenced by cryptoKeyId.	
Particularities and Limitations	
Currently there is no implementation to verify certificates. However, the API is still available for compatibility reasons.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-19 Crypto_30_LibCv_CertificateVerify

4.3 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
CryIf	CryIf_CallbackNotification
NvM	NvM_SetRamBlockStatus NvM_WriteBlock
VStdLib	VStdLib_MemCpy VStdLib_MemClr VStdLib_MemSet
vSecPrim	Several APIs which start with esl_

Table 4-20 Services used by the CRYPTO

4.4 Configurable Interfaces

4.4.1 Callback Functions

This chapter describes the callback functions that are implemented by the CRYPTO and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `Crypto_30_LibCv_Cfg.h` by the CRYPTO.

4.4.1.1 Crypto_30_LibCv_NvBlock_Init_<NvBlock>

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_NvBlock_Init_<NvBlock>`** `(void)`	
Parameter	
void	none
Return code	
Std_ReturnType	E_OK Data initialized.
Std_ReturnType	E_NOT_OK Any error occurred.
Functional Description	
Block specific callback routine which is called by NvM in order to let the Crypto driver copy default data to a RAM block.	
Particularities and Limitations	
-	
Call context	
FunctionCallcontext > This function is Synchronous > This function is Non-Reentrant	

Table 4-21 Crypto_30_LibCv_NvBlock_Init_<NvBlock>

4.4.1.2 Crypto_30_LibCv_NvBlock_ReadFrom_<NvBlock>

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_NvBlock_ReadFrom_<NvBlock>`** `(const void *NvMBuffer)`	
Parameter	
NvMBuffer [in]	RAM mirror where RAM block data can be read from.
Return code	
Std_ReturnType	E_OK Data was copied from buffer.
Std_ReturnType	E_NOT_OK Data was not copied.
Functional Description	
Block specific callback routine which is called by NvM in order to let the Crypto driver copy data from NvM RAM mirror to Crypto RAM block.	

Particularities and Limitations
-
Call context
FunctionCallcontext > This function is Synchronous > This function is Non-Reentrant

Table 4-22 Crypto_30_LibCv_NvBlock_ReadFrom_<NvBlock>

4.4.1.3 Crypto_30_LibCv_NvBlock_WriteTo_<NvBlock>

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_NvBlock_WriteTo_<NvBlock>`** `(void *NvMBuffer)`	
Parameter	
NvMBuffer [out]	RAM mirror where RAM block data shall be written to.
Return code	
Std_ReturnType	E_OK Data was copied to buffer.
Std_ReturnType	E_NOT_OK No data was copied.
Functional Description	
Block specific callback routine which is called by NvM in order to let the Crypto driver copy data from RAM block to NvM RAM mirror.	
Particularities and Limitations	
-	
Call context	
FunctionCallcontext > This function is Synchronous > This function is Non-Reentrant	

Table 4-23 Crypto_30_LibCv_NvBlock_WriteTo_<NvBlock>

4.4.1.4 Crypto_30_LibCv_NvBlock_Callback_<NvBlock>

Prototype	
`Std_ReturnType` **`Crypto_30_LibCv_NvBlock_Callback_<NvBlock>`** `(NvM_ServiceIdType ServiceId, NvM_RequestResultType JobResult)`	
Parameter	
uint8 [in]	The service identifier of the completed request.
NvM_RequestResultType [in]	Result of the single block job.

Return code	
Std_ReturnType	E_OK Callback function has been processed successfully.
Std_ReturnType	E_NOT_OK Callback function has not been processed successfully.
Functional Description	
Block specific callback routine which is called by NvM in order to notify the Crypto driver that an asynchronous single block request has been finished.	
Particularities and Limitations	
-	
Call context	
FunctionCallcontext > This function is Synchronous > This function is Non-Reentrant	

Table 4-24 Crypto_30_LibCv_NvBlock_Callback_<NvBlock>

4.4.1.5 esl_getBytesRNG

Prototype	
`eslt_ErrorCode` **`esl_getBytesRNG`** `(const eslt_Length target_length, eslt_Byte* 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	
This function generates random numbers for primitives which requires random numbers for algorithm execution.	
Particularities and Limitations	
Available if a Default Random Source is configured CRYPTO_30_LIBCV_DEFAULT_RANDOM_SOURCE == STD_ON	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-25 esl_getBytesRNG

4.4.2 Callout Functions

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

4.4.2.1 <WatchdogFuntion>

Prototype	
`void` **`<WatchdogFuntion>`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
This function is called to trigger the Watchdog.	
Particularities and Limitations	
-	
Call context	
FunctionCallcontext > This function is Synchronous > This function is Non-Reentrant	

Table 4-26 <WatchdogFuntion>

4.4.2.2 Appl_Crypto_30_LibCv_SaveContextCallout

Prototype	

`Std_ReturnType` **`Appl_Crypto_30_LibCv_SaveContextCallout`** `(`
`  uint32 objectId,`
`  uint32 jobId,`
`  const uint8 *context,`
`  uint32 contextLength,`
`  uint8 *outputPtr,`
`  uint32 *outputLengthPtr)`	

Parameter	
objectId[in]	Holds the identifier of the Crypto Driver Object.
jobId[in]	Holds the identifier of the Job.
context[in]	Holds the Address of the source context data.
contextLength[in]	Holds the length of the source context.
outputPtr[out]	Pointer to output buffer
outputLengthPtr[in,out]	Pointer to output length buffer
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
Functional Description	
The customer can choose whether the context data is to be written encrypted or in plain text to the output buffer. The outputLengthPtr must be set to the written length. The JobId and ObjectId can be used to check whether the data originates from the correct job and object.	
Particularities and Limitations	
-	
Call context	
FunctionCallcontext > This function is Synchronous > This function is Reentrant	

Table 4-27 Appl_Crypto_30_LibCv_SaveContextCallout

4.4.2.3 Appl_Crypto_30_LibCv_RestoreContextCallout

Prototype	

`Std_ReturnType` **`Appl_Crypto_30_LibCv_RestoreContextCallout`** `(`
`uint32 objectId,`
`uint32 jobId,`
`uint8 *context,`
`uint32 contextLength,`
`const uint8 *inputPtr,`
`uint32 inputLength)`	

Parameter	
objectId[in]	Holds the identifier of the Crypto Driver Object.
jobId[in]	Holds the identifier of the Job.
context[out]	Holds the Address of the destination context data.
contextLength[in]	Holds the length of the destination context.
inputPtr[in]	Pointer to source input buffer.
inputLength[in]	Holds the input buffer length.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
Functional Description	
The customer can choose to write the data encrypted or in plain text to the destination context address. The data shall only be copied if the plain context matches the contextLength. If the context data is encrypted, it must be decrypted first and then written to the workspace address. The JobId and ObjectId can be used to check whether the data originates from the correct job and object.	
Particularities and Limitations	
-	
Call context	
FunctionCallcontext > This function is Synchronous > This function is Reentrant	

Table 4-28 Appl_Crypto_30_LibCv_RestoreContextCallout

4.4.2.4 <Appl_KeyValueChangedCallout>

Prototype	
`void <Appl_KeyValueChangedCallout> ( uint32 cryptoKeyId, uint32 keyElementId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which value was changed.
keyElementId [in]	Holds the identifier of the key element which value has changed.
Return code	
void	none
Functional Description	
This function is called by the LibCv to notify the application when a key value was changed. This function is only available if the callout name is set, otherwise it is disabled. It notifies which key element of the key was changed. The limitations are documented in the chapter 2.12.6.	
Particularities and Limitations	
If the keys are set to their initial value during start-up or initialization of the NvM blocks, the callout will not be called. During the Key value changed callout, the key is still locked for the pending request and cannot be used.	
Call context	
FunctionCallcontext > This function is Synchronous > This function is Reentrant	

Table 4-29 <Appl_KeyValueChangedCallout>

4.4.2.5 <Appl_KeyValiditySetCallout>

Prototype	
`void <Appl_KeyValiditySetCallout> ( uint32 cryptoKeyId, boolean validity)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which validity was changed.
validity [in]	Contains the information which validity was set. True stands for valid and False for invalid.
Return code	
void	none

Functional Description
This function is called by the LibCv to notify the application when the validity of the key has changed. This function is only available if the callout name is set, otherwise it is disabled. The limitations are documented in the chapter 2.12.6
Particularities and Limitations
The callout is not called if the validity of a key has changed when a key is set. If the keys are set to their initial value during start-up or initialization of the NvM blocks, the callout will not be called. During the Key validity changed callout, the key is still locked for the pending request and cannot be used.
Call context
FunctionCallcontext This function is Synchronous This function is Reentrant

Table 4-30 <Appl_KeyValiditySetCallout>

5 Configuration

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

Configuration in DaVinci Configurator 5 Pro

5.1 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.

5.2 NvM Block Needs

The following table includes the configuration constraint for the used NvM blocks.

Parameter	Block Type	Expected Value
NvMBlockUseSetRamBlockStatus	-	TRUE
NvMBlockUseSyncMechanism	-	TRUE
NvMInitBlockCallback	-	Init Block Callback
NvMReadRamBlockFromNvCallback	-	Read from Block Callback
NvMSelectBlockForReadAll	-	TRUE
NvMSelectBlockForWriteAll	DEFERRED	TRUE
	IMMEDIATE	FALSE
NvMSetRamBlockStatusApi	-	TRUE
NvMSingleBlockCallback	-	Block Callback
NvMWriteRamBlockToNvCallback	-	Write to Block Callback
MICROSAR/NvMInvokeCallbacksForWriteAll	-	TRUE
MICROSAR/NvMUseInitCallback	-	TRUE
MICROSAR/NvMUseJobendCallback	-	TRUE

Table 5-1 NvM Block Needs

6 Cybersecurity

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

6.1 Configuration

6.1.1 CMI-CRYPTO-ReadAccess

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

The user of MICROSAR Classic shall restrict read access to cryptographic material according to their cybersecurity concept.

Rationale:

The restriction of access rights to cryptographic material leads to a smaller attack surface to read confidential data.

6.1.2 CMI-CRYPTO-WriteAccess

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

The user of MICROSAR Classic shall restrict write access to cryptographic material according to their cybersecurity concept.

Rationale:

The restriction of access rights to cryptographic material leads to a smaller attack surface to tamper with cryptographic material.

6.1.3 CMI-CRYPTO-NvMConsistencyLevel

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

The user of MICROSAR Classic shall use consistency level NV_CONSISTENCY_LEVEL_DETECT for all NvM blocks.

Rationale:

The access rights of a key element can be changed by a firmware update through a changed configuration. By setting the consistency level to NV_CONSISTENCY_LEVEL_DETECT, such a change is detected, and the already existing key element is replaced by the newly configured initial key element value. This prevents the existing key, which originally had different access rights, from being used with the modified access rights.

6.1.4 CMI-CRYPTO-KeyNotUsed

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

If a key element is not used or not used anymore, the user of MICROSAR Classic shall restrict the access to RA_DENIED and WA_DENIED.

Rationale:

A restriction of the access prevents both compromising and accidental use of this key.

6.1.5 CMI-CRYPTO-SheKeyReadAccess

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial

The user of MICROSAR Classic shall set the read access of SHE key elements to RA_ENCRYPTED.

Rationale:

Without encryption, the key content could be read as plaintext and thus violate confidentially of cryptographic material.

6.1.6 CMI-CRYPTO-SheRamKeyReadAccess

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial

The user of MICROSAR Classic shall set the read access of SHE RAM key elements to RA_ENCRYPTED.

Rationale:

Without encryption, the key content could be read as plaintext and thus violate confidentially of cryptographic material.

6.1.7 CMI-CRYPTO-SheKeyWriteAccess

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial

The user of MICROSAR Classic shall set the write access of SHE key elements to WA_ENCRYPTED. This does not apply to SHE RAM key elements (see 2.4.8).

Rationale:

Without authentication, the key content could be written unauthorized. This could lead to loss of confidentially, integrity, and authenticity of cryptographic material.

6.1.8 CMI-CRYPTO-SheDebugCmd

Technical Cybersecurity Requirements: TCR-MSRC-AccessCryptoMaterial

The user of MICROSAR Classic shall disable SHE CMD_DEBUG.

Rationale:

By disabling the SHE CMD_DEBUG, the attack surface is decreased. Resetting all keys to their initial states by this command would lead to a violation of integrity of cryptographic material. In addition, if the initial values are known, the contents of all keys would no longer be secret.

6.1.9 CMI-CRYPTO-ContextNotUsed

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

The user of MICROSAR Classic shall set Primitive Support Context to false if Save and Restore Context is not used.

Rationale:

By disabling the Primitive Support Context, the attack surface is decreased to read or manipulate cryptographic material.

6.2 Runtime Interfaces: Application

6.2.1 CMI-CRYPTO-ContextCallout

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

The user of MICROSAR Classic shall ensure the confidentiality, integrity, and authenticity of the saved context data when using the callout functions Appl_Crypto_30_LibCv_SaveContextCallout and Appl_Crypto_30_LibCv_RestoreContextCallout.

Rationale:

The saved context may contain confidential cryptographic material that could be read or manipulated if stored in unprotected memory.

6.2.2 CMI-CRYPTO-RandomSeed

Technical Cybersecurity Requirements: TCR-MSRC-ProvideCryptoPrimitives

Before using the functionality Random Number Generation, the user of MICROSAR Classic shall ensure the used key is seeded according to the algorithm. For details, see 2.5.

Rationale:

A random number generator which is not seeded properly may produce predictable data and thus introduce security issues in cryptographic primitives processing the random number generator output.

6.3 Runtime Interfaces: BSW

TCR	Depends on Component(s)	Comment
TCR-MSRC-AccessCryptoMaterial, TCR-MSRC-AccessCertificates	NvM	Needed for reading and writing keys and certificates
TCR-MSRC-ProvideCryptoPrimitives	vSecPrim	Needed for cryptographic primitives

Table 6-1 Cybersecurity-relevant Runtime Interfaces to BSW Components

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
AAD	Additional Authenticated Data
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
ALGOFAM	Algorithm Family
ALGOMODE	Algorithm Mode
ANSI	American National Standards Institute
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CBC	Cipher Block Chaining
CMAC	Cipher-based Message Authentication Code
CMI	Cybersecurity Manual Items
CRT	Chinese Remainder Theorem
CTR	Counter
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DF	Derivation Function
EAD	Embedded Architecture Designer
ECB	Electronic Code Book
ECBD	Elliptic Curve Burmester Desmedt
ECC	Elliptic-curve-cryptography
ECDHE	Elliptic-curve Diffie–Hellman
ECDSA	Elliptic-curve Digital Signature Algorithm
ECU	Electronic Control Unit
EDDSA	Edwards-curve Digital Signature Algorithm
FID	Functional identification

Abbreviation	Description
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
HIS	Hersteller Initiative Software
HMAC	Keyed-Hash Message Authentication Code
ISO	International Organization for Standardization
ISR	Interrupt Service Routine
IV	Initialization Vector
KDF	Key Derivation Function
KE	Key Element
KEA	Key Exchange Algorithm
MAC	Message Authentication Code
MD5	Message-Digest Algorithm 5
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
NIST	National Institute of Standards and Technology
PBKDF	Password-Based Key Derivation Function
PPORT	Provide Port
PRF	Pseudorandom Function
PRNG	Pseudo Random Number Generator
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RNG	Random Number Generator
RPORT	Require Port
RSA	Rivest-Shamir-Adleman
RTE	Runtime Environment
SCC	Smart Charge Communication
SEC	Standards for Efficient Cryptography
SHA	Secure Hash Algorithm
SHE	Secure Hardware Extension
SipHash	short-input PRF Hash
SLH-DSA	Stateless Hash-Based Digital Signature Algorithm
SM	ShangMi (Chinese cryptography standards)
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
TCR	Technical Cybersecurity Requirement
TRNG	True Random Number Generator
UID	Unique identification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com