Elektrobit

EB tresos® AutoCore Generic 8 Crypto documentation

Module release 1.7.26

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Confidential information.

Table of Contents

1. Overview

Welcome to the EB tresos AutoCore Generic 8 Crypto product release notes and documentation.

This document provides:

- Chapter 2, “Crypto module release notes“: details of changes and new features in the current release
- Chapter 3, “Crypto module user's guide “: concept information and configuration instructions
- Chapter 4, “Crypto module references“: configuration parameters and the application programming interface

2. Crypto module release notes

- AUTOSAR R4.3 Rev 0
- AUTOSAR SWS document version: 4.3.0
- Module version: 1.7.26.B337087
- Supplier: Elektrobit Automotive GmbH

2.1. Change log

This chapter lists the changes between different versions.

Module version 1.7.26

2020-06-19

- Changed NO_INIT memory sections to CLEARED
- ASCCRYPTO-2237 Fixed known issue: Crypto does not generate correct symbolic names for CryptoKeyElementIds

Module version 1.7.21

2020-02-21

- Internal module improvement. This module version update does not affect module functionality.

Module version 1.7.20

2020-01-24

- ASCCRYPTO-1605 Fixed known issue: AES-CMAC expanded key can lead to alignment problem

Module version 1.7.19

2019-12-06

- Fixed compiler warning when using GHS compiler

Module version 1.7.16

2019-10-11

- ASCCRYPTO-1504 Fixed known issue: Prefix of compiler abstractions is missing the vendorId and vendorApiInfix
- ASCCRYPTO-1566 Fixed known issue: Callable entities syntax does not match the syntax generated by Rte
- ASCCRYPTO-1228 Fixed known issue: Check for skipping main function can lead to memory exception

Module version 1.7.15

2019-08-09

- ASCCRYPTO-1346 Fixed known issue: Crypto does not generate correct symbolic names for CryptoKeys
- ASCCRYPTO-1314 Fixed known issue: Crypto does not generate correct symbolic names for CryptoDriverObjects
- ASCCRYPTO-1500 Fixed known issue: BSW-MODULE-ENTRY-REF-CONDITIONAL entry for Crypto_MainFunction in Crypto_Bswmd.arxml is not generated
- ASCCRYPTO-1498 Fixed known issue: VENDOR-API-INFIX entry in Crypto_Bswmd.arxml is generated incorrectly

Module version 1.7.11

2019-06-14

- ASCCRYPTO-1243 Fixed known issue: Precalculation for MAC keys (AES-CMAC) during start-up may not be done
- ASCCRYPTO-1297 Fixed known issue: AES-CTRDRBG outputs more data than requested
- ASCCRYPTO-1361 Fixed known issue: ECDSA - failure in random number
- ASCCRYPTO-1290 Fixed known issue: Missing exit of exclusive area if CryptoQueueSize is zero
- The ApiInfix (AI) parameter is now part of the Bswmd.
- ASCCRYPTO-1308 Fixed known issue: Vulnerability to side channel attacks due to not secure 'memcmp' implementation
- ASCCRYPTO-1355 Fixed known issue: AES-ECB ENCRYPT service is not functional

- Improved buffersizes for RSA primitives (PSS, PKCS1V15 and OAEP).

Module version 1.7.9

2019-04-12

- Provided internal cipher AES-ECB externally.
- Improved buffer sizes used for RSA based primitives and changed the feature that symmetric, public and private key sizes can be global configured via Csm.
- Improved Elliptic Curve Diffie-Hellman (ECDH) for x25519 and ECC NIST curves secp256r1 and secp384r1.
- Changed values of reference parameters in XDM and BMD file.
- Removed 'myEcuParameterDefinition' from XDM and BMD file.

Module version 1.7.8

2019-03-22

- Implemented Elliptic Curve Diffie-Hellman (ECDH) for ECC NIST curve secp384r1.

Module version 1.7.7

2019-03-15

- Implemented signature verification and generation using RSASSA-PKCS1V15 with SHA1
- Updated key exchange for selection of either x25519 or NIST ECC elliptic curves.
- Implemented Elliptic Curve Diffie-Hellman (ECDH) for ECC NIST curve secp256r1.

Module version 1.7.6

2019-02-15

- ASCCRYPTO-1085 Fixed known issue: EdDSA and ECDSA SignatureVerify return E_NOT_OK if verification was not successful
- ASCCRYPTO-1053 Fixed known issue: Asynchronous processing can lead to undefined behavior
- Implemented SHA1.
- The certificate, key derivation and key exchange interfaces now reject another call of the same function in parallel.

Module version 1.7.5

2018-12-21

- Improved runtime of AES-GCM, AES-CBC, AES-CFB, RSA-RSAES-OAEP, SHA2, AES-CMAC, SHA2-HMAC, RSA-RSASSA-PKCS1-v1.5 and RSA-RSASSA-PSS
- Implemented ECC NIST Secp256r1 signature generation and signature verification, ECDSA.

Module version 1.7.4

2018-11-09

- Created Tresos Error for 64 Bit support enabled
- Created Tresos Error if no CryptoKeys are configured
- Crypto_ProcessJob() now returns CRYPTO_E_JOB_CANCELED when a synchronous job was cancelled during its processing

Module version 1.7.3

2018-09-21

- ASCCRYPTO-839 Fixed known issue: Compile error in AES_CFB decryption
- ASCCRYPTO-838 Fixed known issue: Compile error in RsaSsa-Pkcs1 v1.5 SIGNATUREGENERATE/VERIFY
- ASCCRYPTO-850 Fixed known issue: Compile error in RSAES-OAEP ENCRYPT
- ASCCRYPTO-855 Fixed known issue: Compile error in Crypto_KeyElementGet()

Module version 1.7.2

2018-09-10

- Implemented Det run-time error types
- Implemented AES-CFB encryption and decryption
- ASCCRYPTO-810 Fixed known issue: Compile error if CryptoQueueSize is zero
- Implemented RSAES-OAEP encryption and decryption
- ASCCRYPTO-812 Fixed known issue: Incorrect use of P2CONST macro may produce compiler errors
- ASCCRYPTO-820 Fixed known issue: Wrong outputPtr in asynchronous SINGLECALL AES-CBC DECRYPT

Module version 1.7.1

2018-07-30

- Disabled the NvM configuration if the variant is AUTOSAR
- Added the parameter CryptoInstanceId. This ID is used to discern several crypto drivers in case more than one driver is used in the same ECU.
- ASCCRYPTO-690 Fixed known issue: Memory access violation in the function Crypto_KeyElementCopy()
- ASCCRYPTO-707 Fixed known issue: Crypto_KeyDerive() writes to an index outside the target key element array
- Improved queue implementation
- Adapted check of target size in KeyElementCopy
- ASCCRYPTO-742 Fixed known issue: Crypto_KeyCopy() results in inconsistent target key and returns wrong value
- Adapted handling of key element read/write access
- ASCCRYPTO-174 Fixed known issue: No calculation of random numbers
- Added empty generation modes "verify" and "verify_swcd" in EB Tresos Studio
- Implemented signature verification and generation using RSASSA-PKCS1V15

Module version 1.7.0

2018-06-07

- ASCCRYPTO-666 Fixed known issue: AES-CMAC: Verification is invalid for key sizes larger than 16 bytes in synchronous single call mode
- Disabled NvM dependencies if there are no persistent key elements
- ASCCRYPTO-667 Fixed known issue: AES-CBC: Encryption and decryption fail for input data sizes of less than 16 bytes in asynchronous single call mode
- Implemented signature verification using RSASSA-PSS
- Implemented Certificate Parse and Certificate Verify
- KeyDerive now supports SHA2-256 as pseudorandom primitive

Module version 1.6.1

2018-05-03

- Adapted queuing in a way that a job can be queued only once. This ensures that a job instance is not overwritten by the following service request for the same job instance.

- ASCCRYPTO-628 Fixed known issue: Wrong DET check in AEADDECRYPT leads to aborted job processing
- ASCCRYPTO-629 Fixed known issue: After calling the KeyDerive function, no further processing of Crypto Driver primitives is possible
- ASCCRYPTO-633 Fixed known issue: Asynchronous GCM with mode STREAMSTART or SINGLECALL locks Crypto Driver Object
- ASCCRYPTO-646 Fixed known issue: GCM decryption only works correctly if length of authentication tag is 128 Bits.
- ASCCRYPTO-639 Fixed known issue: GCM only works correctly if key length is 128 bits
- ASCCRYPTO-634 Fixed known issue: Crypto_KeyElementSet() writes to an index outside the key element array
- ASCCRYPTO-654 Fixed known issue: Crypto_KeyDerive() writes to an index outside the key element array
- ASCCRYPTO-653 Fixed known issue: Crypto_KeyElementCopy() fails when the target key element is configured for internal copy only
- ASCCRYPTO-657 Fixed known issue: Calculation of maximum key element size is incorrect

Module version 1.6.0

2018-04-11

- ASCCRYPTO-479 Fixed known issue: Crypto_ECDHKeyExchangeCalcSecret cannot return E_OK
- ASCCRYPTO-473 Fixed known issue: AES-CTRDRBG: Endless loop occurs during synchronous RandomGenerate
- ASCCRYPTO-492 Fixed known issue: Dangling pointers in UPDATE or FINISH in synchronous primitives CBC, SHA, HMAC, SSG or EdDSA
- Updated Crypto_xVIx_xAIx_Bswmd.arxml to handle multiple Crypto Driver Software instances based on Vendor ID and API Infix
- Adapted initialization of keys and key elements during startup: (i) All non-persistent key elements are initialized by their configured init value; (ii) If reading of a persistent key element from NvM failed, and if the key element has not been initialized by NvM, then the key element is initialized by the Crypto Driver; (iii) A key's state is set to valid if none of its persistent key elements has been initialized by the Crypto Driver. If at least one persistent key element was not loaded successfully by the NvM and has been initialized by the Crypto Driver, the corresponding key is set to invalid. (iv) The initialization of key elements now also considers the precalculation of keys for AES CMAC/ECB.
- Adapted storage of persistent key elements to NvM: (i) Crypto_KeyElementSet() no longer calls NvM_WriteBlock() after setting the key element's value; (ii) Crypto_KeyValidSet() calls NvM_WriteBlock() for all persistent key elements after setting the key's state to valid.

- Improved the handling of Csm_SymKeyType, Csm_AsymPrivateKeyType, and Csm_AsymPublicKeyType in Crypto. If they are configured in Csm, they are just typecasted to internal Crypto types. If they do not exist in Csm, they are created internally in Crypto, using the max size of the key element with ID 1 that is configured in the keys.
- Implemented the key management function KeyDerive (KDF)
- ASCCRYPTO-615 Fixed known issue: GCM handles authentication tag incorrectly if plaintext length is a multiple of 16 bytes

Module version 1.5.2

2018-03-06

- Removed NG Generator workaround caused by a EB tresos Studio framework bug
- ASCCRYPTO-432 Fixed known issue: The Crypto module does not compile if only the AEAD decryption is enabled
- ASCCRYPTO-421 Fixed known issue: Random service with algo mode CTRDRBG fails for asynchronous processing
- Corrected replacement of VendorApiInfix when lowercase letters are used
- Improved the Crypto_xVIx_xAIx_AL_KeyCopy function so that no processing takes place when the source key is invalid
- Improved the Crypto_xVIx_xAIx_AL_KeyElementSet function so that it uses the Crypto_xVIx_xAIx_-KeyElementSet function to set the key element instead of copying it manually
- Added NG Generator workaround caused by a EB tresos Studio framework bug, as required for EB tresos Studio version 14.5.1 up to less than 24.0.0.

Module version 1.5.1

2018-02-16

- Added support for the configuration of the KeyManagement function RandomSeed() via the RANDOM_AL-GORITHM key element
- Improved handling of KeyManagement function call for busy Crypto Driver Object
- Improved handling of outputLengthPtr for every primitive

Module version 1.5.0

2018-02-09

- Added support for SHA2-224, SHA2-256, SHA2-384 and SHA2-512 in the same configuration

- ASCCRYPTO-369 Fixed known issue: SIGNATUREGENERATE/VERIFY (EdDSA) does not reset busy pointer in error case
- Implemented the SipHash-2-4 algorithm using 64 bit tag (message digest)
- Added support for multiple configurations of software Crypto Drivers
- Improved run-time of EdDSA generate and verify
- Implemented the KeyManagement functions Crypto_KeyElementCopy() and Crypto_KeyCopy()
- Implemented the optimized version of the AES CMAC/ECB using precalculated expanded keys, K1 and K2, which are stored in respective key elements

Module version 1.4.0

2017-12-20

- ASCCRYPTO-312 Fixed known issue: Only one crypto primitive for a crypto service is supported
- Added support for multi-instantiation of hardware and software Crypto Drivers

Module version 1.3.0

2017-12-12

- ASCCRYPTO-283 Fixed known issue: Crypto module generates macros for the RamBlockDataAddresses with & as a starting character which leads to compile errors
- ASCCRYPTO-283 Fixed known issue: Crypto does not declare the key data variables to be used in other modules scope
- Improved AES state machine regarding asynchronous/synchronous handling
- Implemented the HMAC algorithm
- Implemented the AEAD with GCM mode
- ASCCRYPTO-313 Fixed known issue: KeyExchange (ECDH) uses wrong KeyElements

Module version 1.2.2

2017-11-27

- ASCCRYPTO-258 Fixed known issue: If CryptoKeyElementSize was equal to the number of provided values in CryptoKeyElementInitValue, the preprocessor derivative resulting from CryptoKeyElementInitValue during code generation had a missing line splice character. This lead to a compilation error.
- ASCCRYPTO-264 Fixed known issue: Crypto module generates duplicate identifiers for NvMRam block and hence the user cannot use the identifier for accessing the NvMRam block

- ASCCRYPTO-255 Fixed known issue: CMAC verification interprets MAC length as bytes instead of BITS
- ASCCRYPTO-293 Fixed known issue: SIGNATUREGENERATE/VERIFY (EdDSA) changes order of bytes in key

Module version 1.2.1

2017-10-20

- ASCCRYPTO-211 The recommended configuration of the Crypto Driver contains all key elements which are specified in SWS_Csm_01022 and not already present in the pre-configuration
- ASCCRYPTO-200 Fixed known issue: Crypto mainfunction needs to be scheduled even if no async job is configured
- ASCCRYPTO-45 Implemented CTR_DRBG based on AES counter mode (AES CTR_DRBG)
- ASCCRYPTO-244 Fixed known issue: CryptoKeyElementInitValue is parsed as comma-separated byte values given in hexadecimal representation (uint8 array). If initialization value contains less hexadecimal values than configured by CryptoKeyElementSize the array is filled with 0x00 in the end.
- ASCCRYPTO-175 Fixed known issue: Crypto primitives could not be called using CRYPTO_OPERATION-MODE_SINGLECALL and asynchronous processing type

Module version 1.2.0

2017-08-30

- ASCCRYPTO-55 Implemented Elliptic Curve Diffie-Hellman (ECDH)
- ASCCRYPTO-55 Fixed known issue: SHA2-512 and Curve25519 (EdDSA) are included
- ASCCRYPTO-152 Fixed known issue: Key elements can be configured as persistent
- Fixed known issue: Key element initial values are fixed
- ASCCRYPTO-151 Fixed known issue: Initialization of Crypto_SymKeyType is fixed
- Fixed known issue: The compiler error caused by a mismatch in parameter Crypto_Generic_Callback was corrected
- ASCCRYPTO-12 Fixed known issue: Variable assignment was corrected

Module version 1.1.0

2017-08-09

- Added NvM support for KeyManagement

Module version 1.0.0

2017-07-28

- Implemented generic part of the Crypto Driver

2.2. New features

- No new features have been added since the last release.

2.3. EB-specific enhancements

This chapter lists the enhancements provided by the module.

- This module provides no EB-specific enhancements.

2.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- Check of key element Id not applicable

  Description:

  - Crypto_KeyElementGet checks key element and returns its index if present.

  Rationale:

  - This requirement is not applicable. It conflicts with SWS_Crypto_00140. The function Crypto_KeyElementGet returns CRYPTO_E_KEY_NOT_AVAILABLE and shall additionally report the runtime error CRYPTO_E_RE_KEY_NOT_AVAILABLE to the DET, if development errer detection is enabled. Also see Bugzilla entry https://bugzilla.autosar.org/show_bug.cgi?id=81704.

  Requirements:

  - SWS_Crypto_00087
- The requirement is obsolete

  Description:

  - Crypto_KeyElementGet has to check the value pointed by resultLPtr.

Rationale:

- This requirement is obsolete in AUTOSAR Specification of Crypto Driver, 4.3.1.

Requirements:

- SWS_Crypto_00093

- The requirement is obsolete

Description:

- Crypto_KeyElementIdsGet has to check the value pointed by keyElementIdsPtr.

Rationale:

- This requirement is obsolete in AUTOSAR Specification of Crypto Driver, 4.3.1.

Requirements:

- SWS_Crypto_00164

- Support input lengths of size zero

Description:

- Some algorithms (e.g. GCM) can produce valid output, even if the input size is zero.

Rationale:

- Implementation is consistent with Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=81483

Requirements:

- SWS_Crypto_00142

- Only single call processing for RandomGenerate

Description:

- For the RandomGenerate service, Crypto_ProcessJob() only supports the operation mode CRYPTO_OPERATIONMODE_SINGLECALL.

Rationale:

- Implementation is consistent with R4.4, see also Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=78133

Requirements:

- SWS_Crypto_91003

- No support for CryptoKeyElementVirtualTargetRef

  Description:

  - The Crypto Driver does not support virtual key elements (CryptoKeyElementVirtualTargetRef).

  Rationale:

  - Currently there is no user request regarding virtual key elements. The SWS is currently under discussion (also see http://www.autosar.org/bugzilla/show_bug.cgi?id=80027).

  Requirements:

  - ECUC_Crypto_00028

- It is not possible to cancel every job directly.

  Description:

  - Crypto_CancelJob() should not wait until cancelation of the job is possible.

  Rationale:

  - This requirement is not applicable. Crypto_CancelJob() should not block the application until cancelation is possible.

  Requirements:

  - SWS_Crypto_00143

- The Crypto_CancelJob() API is inconsistent.

  Description:

  - Crypto_CancelJob() expects the input parameter Crypto_JobType* job instead of Crypto_JobInfoType* job.

  Rationale:

  - This requirement is not applicable. Crypto_JobInfoType* job is only available as a const pointer. It is replaced by SWS_Crypto_00122_CORRECTION.

  Requirements:

  - SWS_Crypto_00122

- Queue requirement only applicable for asynchrounous jobs.

  Description:

- Crypto_ProcessJob() can only check a queue and return CRYPTO_E_QUEUE_FULL if the job is asynchronous.

Rationale:

- This requirement is not applicable. CRYPTO_E_QUEUE_FULL should only be returned if an asynchronous job is passed to Crypto_ProcessJob. It is replaced by SWS_Crypto_00032_CORRECTION.

Requirements:

- SWS_Crypto_00032

- Synchronous job will not wait if Crypto Driver Object is busy.

Description:

- If Crypto_ProcessJob() waits while the crypto driver object is busy before processing a synchronous job, the application might be blocked.

Rationale:

- This requirement is not applicable. It conflicts with SWS_Crypto_00034. A synchronous job shall be rejected if the crypto driver object is busy. Also see Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=77372.

Requirements:

- SWS_Crypto_00120

- Reference SWS_Crypto_00044 does not exist

Description:

- The index of the different key elements from the different crypto services has a wrong reference.

Rationale:

- This requirement is not applicable. Referenced requirement SWS_Crypto_00044 does not exist in specification.

Requirements:

- SWS_Crypto_00037

- Return value CRYPTO_E_KEY_INVALID does not exist

Description:

- If a key is in the state "invalid", crypto services which make use of that key, cannot return CRYPTO_E_KEY_INVALID.

Rationale:

- This requirement is not applicable. CRYPTO_E_KEY_INVALID is not defined. CRYPTO_E_KEY_-NOT_VALID shall be returned. It's replaced by requirement SWS_Crypto_00039_CORRECTION.

Requirements:

- SWS_Crypto_00039

- The parameter versionInfo has to be an out parameter

Description:

- Service name Crypto_GetVersionInfo: parameter versionInfo has to be an out parameter.

Rationale:

- This requirement is not applicable. The parameter versionInfo has to be an out parameter. It's replaced by requirement SWS_Crypto_91001_CORRECTION.

Requirements:

- SWS_Crypto_91001

- Check of key element buffer not applicable

Description:

- Crypto_KeyElementGet has to check the value pointed by resultLengthPtr.

Rationale:

- This requirement is not applicable. It conflicts with SWS_Crypto_00093 as the only way to check if the buffer is sufficient to store the result is to check the value pointed by resultLengthPtr. Also see Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=77804.

Requirements:

- SWS_Crypto_00147

- The parameter keyElementIdsLengthPtr has to be an inout parameter

Description:

- Service name Crypto_KeyElementIdsGet: keyElementIdsLengthPtr has to be an inout parameter.

Rationale:

- This requirement is not applicable. The parameter keyElementIdsLengthPtr has to be an inout parameter. It's replaced by requirement SWS_Crypto_00160_CORRECTION.

Requirements:

- SWS_Crypto_00160

- Check of buffer keyElementIds not applicable

Description:

- Crypto_KeyElementIdsGet has to check the value pointed by resultLengthPtr.

Rationale:

- This requirement is not applicable. It conflicts with SWS_Crypto_00093 as the only way to check if the buffer is sufficient to store the result is to check the value pointed by resultLengthPtr. Also see Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=77804.

Requirements:

- SWS_Crypto_00163

- The parameter seedPtr does not exist

Description:

- Crypto_RandomSeed cannot check seedPtr.

Rationale:

- This requirement is not applicable. The parameter seedPtr does not exist. It's replaced by requirement SWS_Crypto_00130_CORRECTION.

Requirements:

- SWS_Crypto_00130

- The parameter seedLength does not exist

Description:

- Crypto_RandomSeed cannot check seedLength.

Rationale:

- This requirement is not applicable. The parameter seedLength does not exist. It's replaced by requirement SWS_Crypto_00131_CORRECTION.

Requirements:

- SWS_Crypto_00131

- The parameter pubValueLengthPtr does not exist

Description:

- Crypto_KeyExchangeCalcPubVal cannot use pubValueLengthPtr.

Rationale:

- This requirement is not applicable. Parameter pubValueLengthPtr does not exist, rename to publicValueLengthPtr. It's replaced by requirement SWS_Crypto_00106_CORRECTION.

Requirements:

- SWS_Crypto_00106

- The parameter pubValueLengthPtr does not exist

Description:

- Crypto_KeyExchangeCalcPubVal cannot check pubValueLengthPtr.

Rationale:

- This requirement is not applicable. Parameter pubValueLengthPtr does not exist, rename to publicValueLengthPtr. It's replaced by requirement SWS_Crypto_00107_CORRECTION.

Requirements:

- SWS_Crypto_00107

- The parameter partnerPubValueLength does not exist

Description:

- Crypto_KeyExchangeCalcSecret cannot check partnerPubValueLength.

Rationale:

- This requirement is not applicable. Parameter partnerPubValueLength does not exist, rename to partnerPublicValueLength. It's replaced by requirement SWS_Crypto_00115_CORRECTION.

Requirements:

- SWS_Crypto_00115

- The parameter validateCryptoKeyId does not exist

Description:

- If the parameter validateCryptoKeyId is out of range and if default error detection for the Crypto Driver is enabled, the function Crypto_CertificateVerify shall report CRYPTO_E_PARAM_HANDLE to the DET and return E_NOT_OK.

Rationale:

- This requirement is not applicable. The parameter validateCryptoKeyId does not exist. It's replaced by requirement SWS_Crypto_00174_CORRECTION.

Requirements:

- SWS_Crypto_00174

- CryptoDriverObjectId does not start from zero

Description:

- CryptoDriverObjectId shall be consecutive, gapless and shall start from zero.

Rationale:

- This requirement is not applicable. It's invalidated by note 'The Ids in the configuration containers shall be consecutive, gapless and shall start from zero'. It's replaced by requirement ECUC_Crypto_00009_CORRECTION.

Requirements:

- ECUC_Crypto_00009

- CryptoPrimitiveRef has wrong multiplicity

Description:

- CryptoPrimitiveRef shall have multiplicity 1..*.

Rationale:

- This requirement is not applicable. The multiplicity 1 prevents the configuration of one single driver object which uses all primitives. It's replaced by requirement ECUC_Crypto_00018_CORRECTION. Also see Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=77578.

Requirements:

- ECUC_Crypto_00018

- CryptoKeyId does not start from zero

Description:

- CryptoKeyId shall be consecutive, gapless and shall start from zero.

Rationale:

- This requirement is not applicable. It's invalidated by note 'The Ids in the configuration containers shall be consecutive, gapless and shall start from zero'. It's replaced by requirement ECUC_Crypto_00012_CORRECTION.

Requirements:

- ECUC_Crypto_00012

- CryptoKeyElementId does not start from zero

Description:

- CryptoKeyElementId shall be consecutive, gapless and shall start from zero.

Rationale:

- This requirement is not applicable. Identifier shall have a range starting with 0. It's replaced by requirement ECUC_Crypto_00021_CORRECTION.

Requirements:

- ECUC_Crypto_00021

- Crypto_KeyElementGet returns E_NOT_OK if the key is invalid

Description:

- Crypto_KeyElementGet will return E_NOT_OK instead of E_OK if the key is invalid without calling the lower layer functions.

Rationale:

- The key is already in use. See https://bugzilla.autosar.org/show_bug.cgi?id=79359

Requirements:

- SWS_Crypto_91006

- Unclear specification about result length configuration

Description:

- The HASH and MACGENERATE services shall truncate the result depending on the configured job->jobPrimitiveInfo->primitiveInfo->resultLength.

Rationale:

- This requirement is not applicable. It does not explain the relationship between the configured result length and the outputLength API parameter. Further, the Csm has no requirement that specifies which

value shall be stored to the job data structure. Also see Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=81356.

Requirements:

- SWS_Crypto_00065

- Corrected handling of CRYPTO_E_CANCELED

Description:

- If the cancelled job is synchronous, Crypto_ProcessJob() shall return CRYPTO_E_CANCELED.

Rationale:

- The requirement has been corrected in R4.3.1 as the wrong function has been named. The return value CRYPTO_E_JOB_CANCELED shall be returned by Crypto_ProcessJob() and not by Crypto_CancelJob(). When Crypto_CancelJob() returns CRYPTO_E_JOB_CANCELED, this has a different meaning, namely that the cancellation had to be postponed. Also see Bugzilla entry https://bugzilla.autosar.org/show_bug.cgi?id=77374.

Requirements:

- SWS_Crypto_00144

- It is not possible to report CRYPTO_E_INIT_FAILED to the DET

Description:

- The initialization of the Crypto Driver can not fail.

Rationale:

- This requirement is not applicable. The Crypto_Init function can not report CRYPTO_E_INIT_FAILED to the DET.

Requirements:

- SWS_Crypto_00045

- Key generation functionality not supported.

Description:

- The key generation functionality is not supported by this CRypto driver.

Rationale:

- This functionality has not yet been requested or needed in any senario.

Requirements:

- EB_Crypto_00082

- No support for CryptoKeyElementReadAccess CRYPTO_RA_ENCRYPTED and CryptoKeyElementWriteAccess CRYPTO_WA_ENCRYPTED

  Description:

  - The Crypto Driver does not support encrypted keys. The ENCRYPTED CryptoKeyElementReadAccess and CryptoKeyElementWriteAccess have to be disabled.

  Rationale:

  - These requirements are not applicable because the Crypto Driver does not support encrypted keys. Setting the access of a key element to ENCRYPTED would lead to an invalid configuration.

  Requirements:

  - ECUC_Crypto_00024, ECUC_Crypto_00027

- The requirement is only relevant for crypto hardware.

  Description:

  - Key Management Interface: There is no underlying crypto hardware available to be checked.

  Rationale:

  - This requirement is not applicable. There is no underlying crypto hardware for SW Crypto drivers.

  Requirements:

  - SWS_Crypto_00145

- Key management functions cannot return CRYPTO_E_BUSY

  Description:

  - Key management functions will not return CRYPTO_E_BUSY as specified in return values of key management functions.

  Rationale:

  - The key management functions are not linked to any driver object and therefore cannot check their current state.

  Requirements:

  - SWS_Crypto_00148, SWS_Crypto_00155, SWS_Crypto_00160, SWS_Crypto_91007, SWS_Crypto_91009, SWS_Crypto_91010, SWS_Crypto_91011, SWS_Crypto_00171

- Key generation is not supported

  Description:

  - New keys cannot be generated via Crypto_KeyGenerate. The function returns E_NOT_OK if all error checks succeeded.

  Rationale:

  - The hardware does not provide a key generation functionality.

  Requirements:

  - SWS_Crypto_00165

2.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- The Crypto Driver only supports Certificate Parse based on self-descriptive card verifiable (CV) and Certificate Verify using RSA-SSA-PSS described in PKCS #1 v2.2: RSA Cryptography Standard 2012.
- Key generation interface: The function of the key generation interface Crypto_KeyGenerate() is not supported.
- Random service: The random service is only functional when used with a Csm module of version 3.0.-2 or later. This is caused by an unclear handling in the Csm and Crypto AUTOSAR specification (see AUTOSAR Bugzilla RfC #78133).
- The functionality of the following configuration element of the CryptoKeyElement is not supported:
  - CryptoKeyElementVirtualTargetRef
- Crypto Driver Object: This delivery is based on one single Crypto Driver Object.
- Rejected AUTOSAR SWS Requirements: Multiple requirements were rejected due to errors. Requirements: SWS_Crypto_00037, SWS_Crypto_00039, SWS_Crypto_91001, SWS_Crypto_00140, SWS_Crypto_00139, SWS_Crypto_00160, SWS_Crypto_00130, SWS_Crypto_00131, SWS_Crypto_00106, SWS_Crypto_00107, SWS_Crypto_00115, SWS_Crypto_00174, ECUC_Crypto_00009, ECUC_Crypto_00018, ECUC_Crypto_00012, SWS_Crypto_00120, SWS_Crypto_00147, SWS_Crypto_00163
- Reporting of undefined DET errors: For DET errors CRYPTO_E_RE_SMALL_BUFFER, CRYPTO_E_RE_ENTROPY_EXHAUSTED, CRYPTO_E_RE_KEY_NOT_AVALIABLE and CRYPTO_E_RE_KEY_EXTRACT_DENIED, the crypto module currently reports the error codes listed in SWS_Crypto_00043 according to the following mapping: CRYPTO_E_RE_SMALL_BUFFER => CRYPTO_E_SMALL_BUFFER; CRYPTO_E_RE_ENTROPY_EXHAUSTED => CRYPTO_E_ENTROPY_EX-

HAUSTION; CRYPTO_E_RE_KEY_NOT_AVALIABLE => CRYPTO_E_KEY_NOT_AVAILABLE; CRYPTO_E_RE_KEY_EXTRACT_DENIED => CRYPTO_E_KEY_READ_FAIL

- Multi-instantiation: The multi-instantiation of Hardware and Software Crypto Drivers is only functional when used with a CryIf module of version 1.0.3 or later.
- IV for AEAD with GCM: AEAD with GCM supports only initialization vectors of 96 bits length.
- The configured sizes of CsmSymKeyMaxLength, CsmAsymPrivateKeyMaxLength and CsmAsymPublicKeyMaxLength in the Csm module only affect Crypto if used with a Csm module of version 3.0.8 or later.
- Key derivation interface: The key derivation interface is only functional when used with a CryIf module of version 1.0.6 or later.
- RSAES-OAEP: Due to missing enumeration values in CsmDecryptAlgorithmSecondaryFamily and CsmEncryptAlgorithmSecondaryFamily for hash algorithms in the Csm AUTOSAR specification, it is not possible to configure the hash algorithm used for the RSAES-OAEP encryption and decryption services properly (see also AUTOSAR Bugzilla RfC #81809). Therefore, CsmDecryptAlgorithmSecondaryFamily and CsmEncryptAlgorithmSecondaryFamily should be configured to CRYPTO_ALGOFAM_NOT_SET. The Crypto Driver selects the hash algorithm configured in the appropriate RSAES-OAEP encryption and decryption primitives in the Crypto Driver Object. As a consequence, this allows only one RSAES-OAEP encryption and one RSAES-OAEP decryption primitive.
- SHA1 Primitive: the implementation of SHA1 supports only input lengths < 2^32 bytes.

2.6. Open-source software

`Crypto` does not use open-source software.

3. Crypto module user's guide

3.1. Overview

This user's guide describes the concept and the configuration of the Crypto Driver (`Crypto`) module.

The user's guide is intended for readers who have good knowledge of AUTOSAR and about the purpose of the Crypto Driver. The information provided here helps you to integrate the `Crypto` module in an AUTOSAR project.

- Section 3.2, “Background information” explains the basic functionality of `Crypto`.
- Section 3.3, “Cryptographic capabilities” describes the capabilities of the `Crypto` module regarding supported cryptographic algorithms and key management.
- Section 3.4, “Job processing” explains features realized for processing a job.
- Section 3.5, “Configuring the Crypto module” provides information on how to configure `Crypto`.

For `Crypto` parameter descriptions, see Section 4.1, “Configuration parameters”.

3.2. Background information

The AUTOSAR `Crypto` module provides an interface to implemented cryptographic primitives. These cryptographic primitives are either implemented by a dedicated microcontroller hardware, like an HSM, that is accessed by the `Crypto` module. Thus, the Crypto acts as a driver. Or, the primitives are directly implemented in software by the `Crypto` module itself.

NOTE

Cryptographic primitive

A cryptographic primitive is the combination of a cryptographic service (e.g. ENCRYPT, DECRYPT, MAC_GENERATE, SIGNATURE_VERIFY, HASH), an algorithm family (e.g. AES, RSA, SHA3) and an algorithm mode (e.g. ECB, CBC, RSAES_OAEP).

The `Crypto` module is located in the microcontroller abstraction layer of the AUTOSAR stack. It is interfaced via the `CryIf` in the interface layer, which manages the access to the Crypto Driver Objects of eventually multiple `Crypto` modules. In turn, the `CryIf` is interfaced via the `Csm` in the service layer. The `Csm` provides a job-based access to the cryptographic primitives within the `Crypto` modules by applications and basic software modules.

Therefore, the three modules `Csm`, `CryIf`, and `Crypto` have to be configured together to work correctly. To enable the `Csm` to use any cryptographic primitives and keys, they have to be referenced within the `CryIf`

configuration. The CryIf must reference Crypto Driver Objects and Crypto keys. Only the primitives of the referenced Crypto Driver Objects and the referenced Crypto keys can be used by the Csm.

3.2.1. General behavior

A Crypto Driver Object represents an instance of either an independent cryptographic hardware device or a cryptographic software library. The Crypto module currently implements one Crypto Driver Object. The Crypto Driver Object references all configured cryptographic primitives. There is only one workspace for each Crypto Driver Object. Therefore, only one primitive can be performed at a time. The cryptographic requests and results are routed via dedicated channels, which link the specific Crypto Driver Object via the CryIf to the Csm.

The Crypto provides a job interface and a key management interface. The job interface can be used to execute or stop cryptographic primitives while the key management can be used to modify, extract or generate keys necessary for the cryptographic operations.

3.2.2. Multi-instantiation

Different instances of the Crypto module can be used to provide access to different cryptographic software libraries or hardware modules. The different instances are handled by the CryIf module in the AUTOSAR layer above. This affects every EB tresos Studio project containing a Crypto module, also if you do not use multiple Crypto instances. The reason is that the Crypto module can be used in a setting with several Crypto instances.

As described in the AUTOSAR Specification of BSW Module Description Template [3], a Vendor ID and a Vendor API Infix are used to distinguish different instances of an AUTOSAR module. Elektrobit Automotive GmbH uses the number 1 as Vendor ID. Vendor ID and Vendor API Infix are used in the naming of files, functions, data types etc.

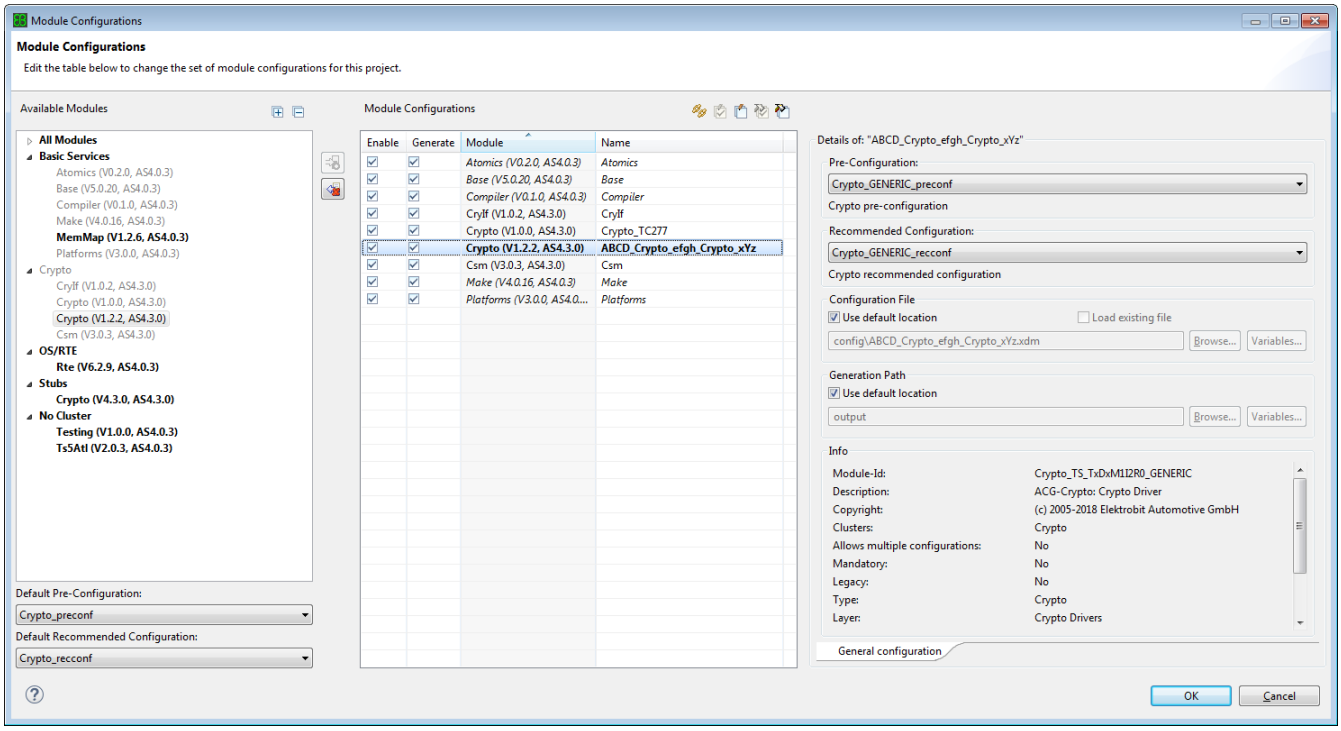

Figure 3.1. Example module configuration with multiple `Crypto` instances

During module configuration, you can add different `Crypto` instances to the same project. As an example, Figure 3.1, “Example module configuration with multiple `Crypto` instances” shows that both the `Crypto` hardware module, named *Crypto_TC277*, and the `Crypto` software module, named *ABCD_Crypto_efgh_Crypto_xYz*, were added. Note that the names of the modules must be unique and that EB tresos Studio, where necessary, creates appropriate module names by appending an ascending number.

You may change the module name. EB tresos Studio uses the module name to create a Vendor API Infix by removing all underscores and the string *Crypto*. The Vendor API Infix is shown on the **Published Information** tab (see example in Figure 3.2, “Location of Vendor API Infix”).

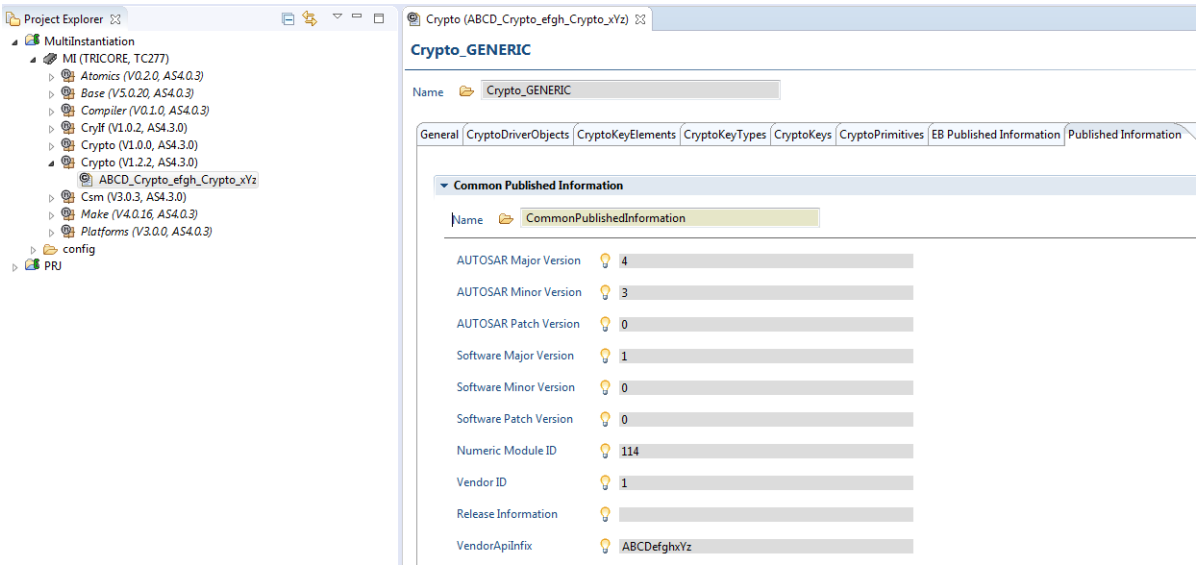

Figure 3.2. Location of Vendor API Infix

To handle multiple instantiation, the implementation of the `Crypto` module uses placeholders in the naming of files, functions, data types etc., e.g. *xVIx_xAIx*. The file names used in the `Crypto` folder start with *Crypto_xVIx_xAIx_*. During the generation of the EB tresos Studio project, the static files are renamed and copied to the *output* folder. The generated folders *output\generated\instance\include* and *output\generated\instance\src* hold the files that were copied from the module's *include* and *src* folders below the EB tresos *plugins* folder. The files that are generated during the generation of the EB tresos Studio project are still created in their output folder as before. At the same time, they are renamed by replacing *xVIx_xAIx* with the actual Vendor ID and Vendor API Infix.

Furthermore, functions, external variables, etc. in the source code are renamed accordingly. Finally, *xVIx* is replaced with the *VendorId* that is generally *1* for Elektrobit Automotive GmbH. And *xAIx* is replaced by the configured *VendorApiInfix*. In the example shown in Figure 3.3, “Example directory structure of generated `Crypto` files”, the *VendorApiInfix* is *EB*, and there is a *Crypto_1_EB* subdirectory below the *instance* folder. The files contained in *include* and *src* are copied from the corresponding module's directories and renamed e.g. from *Crypto_xVIx_xAIx_AL_Common.c* to *Crypto_1_EB_AL_Common.c*.

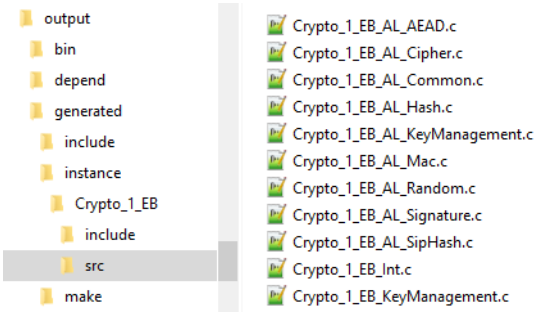

Figure 3.3. Example directory structure of generated `Crypto` files

NOTE 

Vendor API Infix for single Crypto module

If only a single `Crypto` module is present and you accept the default *Name* of the `Crypto` module in the *Module Configurations*, the Vendor API Infix is empty. In this case, the substitution of *xVIx_xAIx* in the naming of files, functions etc. is also an empty string, e.g. *Crypto_AL_Common.c*.

WARNING 

Use of correct file paths for build process

Due to the described adaptations for multiple instances, it is essential that the build process considers the correct directory paths. Do not include the source files from the original module's folder into the compilation and linkage. Instead, use the files generated below *output\generated\instance*. If you use the wrong files, you receive linker errors.

3.3. Cryptographic capabilities

The `Crypto` is delivered with the **Crypto_preconf** preconfiguration. This preconfiguration represents the capabilities of the `Crypto`. The capabilities can be divided into two main topics: supported algorithms and key management.

WARNING 

Key consistency

The AUTOSAR Specification of Crypto Driver [4] says that it "is up to the application to only change key, if currently no primitive works with that key (element)." If a key is modified when the key is currently in use by a service or key management function, the key will probably become inconsistent during its processing.

3.3.1. Supported algorithms

The algorithms supported by the `Crypto` are preconfigured in the Crypto Driver Object. Each primitive that the `Crypto` supports has an entry in the `CryptoPrimitives` container. Each configured primitive represents a cryptographic service with different algorithm families and modes.

The following table shows the provided `Crypto` primitives. The column **2nd Family** refers to the secondary family. **2nd Mode** refers to secondary mode. In some cases, the secondary family is shown in parentheses, e.g. NOT_SET (AES). This means that the family is not defined by the configuration, and is used internally.

Service	Family	Mode	2nd Family	2nd Mode	Key Bitlengths
AEAD_ENCRYPT	AES	GCM	NOT_SET (AES)	ECB	128 - *
AEAD_DECRYPT	AES	GCM	NOT_SET (AES)	ECB	128 - *

Service	Family	Mode	2nd Family	2nd Mode	Key Bitlengths
ENCRYPT	AES	ECB	NOT_SET	-	128
ENCRYPT	AES	ECB	NOT_SET	-	192
ENCRYPT	AES	ECB	NOT_SET	-	256
ENCRYPT	AES	CBC	NOT_SET (AES)	ECB	128
ENCRYPT	AES	CBC	NOT_SET (AES)	ECB	192
ENCRYPT	AES	CBC	NOT_SET (AES)	ECB	256
ENCRYPT	AES	CFB	NOT_SET (AES)	ECB	128
ENCRYPT	AES	CFB	NOT_SET (AES)	ECB	192
ENCRYPT	AES	CFB	NOT_SET (AES)	ECB	256
ENCRYPT	RSA	RSAES_OAEP	SHA2_224	-	*
ENCRYPT	RSA	RSAES_OAEP	SHA2_256	-	*
ENCRYPT	RSA	RSAES_OAEP	SHA2_384	-	*
ENCRYPT	RSA	RSAES_OAEP	SHA2_512	-	*
DECRYPT	AES	ECB	NOT_SET	-	128
DECRYPT	AES	ECB	NOT_SET	-	192
DECRYPT	AES	ECB	NOT_SET	-	256
DECRYPT	AES	CBC	NOT_SET (AES)	ECB	128
DECRYPT	AES	CBC	NOT_SET (AES)	ECB	192
DECRYPT	AES	CBC	NOT_SET (AES)	ECB	256
DECRYPT	AES	CFB	NOT_SET (AES)	ECB	128
DECRYPT	AES	CFB	NOT_SET (AES)	ECB	192
DECRYPT	AES	CFB	NOT_SET (AES)	ECB	256
DECRYPT	RSA	RSAES_OAEP	SHA2_224	-	*
DECRYPT	RSA	RSAES_OAEP	SHA2_256	-	*
DECRYPT	RSA	RSAES_OAEP	SHA2_384	-	*
DECRYPT	RSA	RSAES_OAEP	SHA2_512	-	*
HASH	SHA1	NOT_SET	NOT_SET	-	-
HASH	SHA2_224	NOT_SET	NOT_SET	-	-
HASH	SHA2_256	NOT_SET	NOT_SET	-	-
HASH	SHA2_384	NOT_SET	NOT_SET	-	-
HASH	SHA2_512	NOT_SET	NOT_SET	-	-

Service	Family	Mode	2nd Family	2nd Mode	Key Bitlengths
SIGNATURE_GENERATE	ED25519	NOT_SET	SHA2_512	-	256
SIGNATURE_GENERATE	RSA	RSASSA_PKCS1_v1_5	SHA2_224	-	*
SIGNATURE_GENERATE	RSA	RSASSA_PKCS1_v1_5	SHA2_256	-	*
SIGNATURE_GENERATE	RSA	RSASSA_PKCS1_v1_5	SHA2_384	-	*
SIGNATURE_GENERATE	RSA	RSASSA_PKCS1_v1_5	SHA2_512	-	*
SIGNATURE_GENERATE	ECCNIST	NOT_SET	SHA2_256	-	256
SIGNATURE_VERIFY	RSA	RSASSA_PSS	SHA2_224	-	*
SIGNATURE_VERIFY	RSA	RSASSA_PSS	SHA2_256	-	*
SIGNATURE_VERIFY	RSA	RSASSA_PSS	SHA2_384	-	*
SIGNATURE_VERIFY	RSA	RSASSA_PSS	SHA2_512	-	*
SIGNATURE_VERIFY	RSA	RSASSA_PKCS1_v1_5	SHA2_224	-	*
SIGNATURE_VERIFY	RSA	RSASSA_PKCS1_v1_5	SHA2_256	-	*
SIGNATURE_VERIFY	RSA	RSASSA_PKCS1_v1_5	SHA2_384	-	*
SIGNATURE_VERIFY	RSA	RSASSA_PKCS1_v1_5	SHA2_512	-	*
SIGNATURE_VERIFY	ED25519	NOT_SET	SHA2_512	-	256

Service	Family	Mode	2nd Family	2nd Mode	Key Bitlengths
SIGNATURE_VERIFY	ECCNIST	NOT_SET	SHA2_256	-	512
MAC_GENERATE	AES	CMAC	NOT_SET (AES)	ECB	128
MAC_GENERATE	AES	CMAC	NOT_SET (AES)	ECB	192
MAC_GENERATE	AES	CMAC	NOT_SET (AES)	ECB	256
MAC_GENERATE	SHA2_256	HMAC	NOT_SET (SHA2_256)	-	8 - *
MAC_GENERATE	SIPHASH	SIPHASH_2_4	NOT_SET	-	128
MAC_VERIFY	AES	CMAC	NOT_SET (AES)	ECB	128
MAC_VERIFY	AES	CMAC	NOT_SET (AES)	ECB	192
MAC_VERIFY	AES	CMAC	NOT_SET (AES)	ECB	256
MAC_VERIFY	SHA2_256	HMAC	NOT_SET (SHA2_256)	-	8 - *
MAC_VERIFY	SIPHASH	SIPHASH_2_4	NOT_SET	-	128
RANDOM	RNG	NOT_SET	NOT_SET	-	-
RANDOM	AES	CTRDRBG	NOT_SET (AES)	ECB	256

3.3.2. Key management

In the `Crypto`, key elements (`CryptoKeyElement`) are type definitions including its size, initial values, initialization vectors, etc., as shown in Figure 3.4, “Exemplary key definition”. A key type (`CryptoKeyType`) references at least one `CryptoKeyElement`. A key (`CryptoKey`) references a `CryptoKeyType`, so it is an instance of this key type with instances of corresponding `CryptoKeyElements`. Depending on the service and algorithm for which a key (`CryptoKey`) should be used, the referenced `CryptoKeyType` may contain multiple `CryptoKeyElements`.

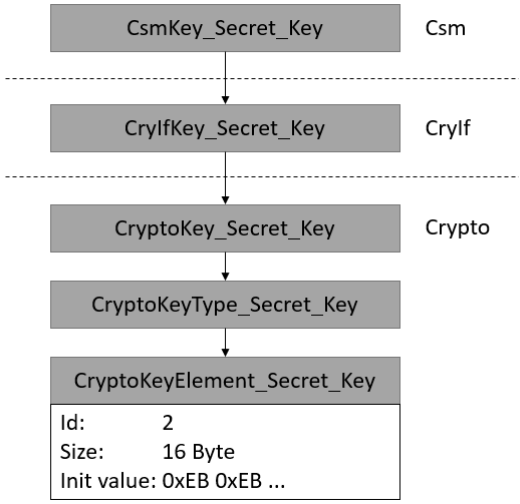

Figure 3.4. Exemplary key definition

Each `CryptoKeyElement` can be used in multiple `CryptoKeyTypes`. Each `CryptoKeyType` can be referenced by multiple `CryptoKeys`.

3.3.2.1. Supported key management functions

The following overview lists the key management functions provided by the `Crypto` and their supported cryptographic algorithms. The column **2nd Family** refers to the secondary family. **2nd Mode** refers to secondary mode. **Algo. selection** specifies the key element that you have to configure to select a particular cryptographic algorithm.

Service	Family	Mode	2nd Family	2nd Mode	Algo. Selection
RandomSeed	AES	ECB	-	-	CRYPTO_KE_-RANDOM_AL-GORITHM
RandomSeed	-	-	-	-	CRYPTO_KE_-RANDOM_AL-GORITHM
KeyDerive	SHA2_256	HMAC	SHA2_256	-	CRYPTO_KE_-KEYDERI-VATION_ALGO-RITHM
KeyDerive	SHA2_256	-	-	-	CRYPTO_KE_-KEYDERI-

Service	Family	Mode	2nd Family	2nd Mode	Algo. Selection
					VATION_ALGO-RITHM
KeyExchange	ED25519	-	-	-	CRYPTO_KE_-KEYEX-CHANGE_AL-GORITHM
KeyExchange	ECCNIST	-	-	-	CRYPTO_KE_-KEYEX-CHANGE_AL-GORITHM, CRYPTO_KE_-KEYEX-CHANGE_CURVE
KeyExchange	ECCNIST	-	-	-	CRYPTO_KE_-KEYEX-CHANGE_AL-GORITHM, CRYPTO_KE_-KEYEX-CHANGE_CURVE
CertificateParse	-	-	-	-	-
CertificateVerify	RSA	RSASSA_PSS	SHA2_256	-	-

NOTE

Result length configuration parameters

The AUTOSAR Specification of Crypto Service Manager [1] specifies configuration parameters with a result length, e.g. `CsmHashResultLength`. A requirement is missing that specifies how to proceed with these result length parameters (see also RfC #81356). According to a `Crypto` requirement, the HASH and MAC_GENERATE services shall truncate the result depending on this parameter. But it does not explain the relationship between the result length configuration parameter and the outputLength API parameter. Further, the remaining `Csm` services are not considered. As a consequence, the `Crypto` does not do any processing regarding the result length configured in the `Crypto Service Manager`.

The following subchapter provides more detailed information on dedicated key management functions.

3.3.2.1.1. Key derivation function (KDF)

The implemented key derivation function is KDF in counter mode. As input parameters, password, salt, and an algorithm as pseudorandom function are used. The key element holding the number of iterations can be

omitted because the number of iterations is given by the size of the target key. The implementation does not use the configuration parameter `CryptoKeyDeriveIterations`. The following algorithms are supported:

Algorithm	Macro name
CryptoPrimitive_SHA256_HMAC_Mac_Generate	CRYPTO_XVIX_XAIX_MACGENERATE_SHA2_-256_HMAC_ALGORITHM
CryptoPrimitive_SHA2_256_Hash	CRYPTO_XVIX_XAIX_HASH_SHA2_256_ALGO-RITHM

The algorithm selected for KDF must be referenced in `CryptoPrimitiveRef` inside the `CryptoDriver-Object`.

3.3.2.1.2. Leap year check in certificate functions

The certificates store the dates in the format YYMMDD. Since only two digits of the year are available, correct checking for leap year is not possible. If both digits of the year are zero, determination of the leap year might be incorrect.

3.3.2.2. Preconfiguration

The **Crypto_preconf** preconfiguration specifies all key types and key elements that the `Crypto` supports. If required, you can extend key types by other key elements and thus create your own key types. For this purpose, the **Crypto_recconf** recommended configuration is provided.

Crypto_preconf together with **Crypto_recconf** contain all key elements that are specified in SWS_Csm_-01022 of the AUTOSAR Specification of Crypto Service Manager [1].

You can view the content of **Crypto_preconf** and **Crypto_recconf** in the corresponding files that are located in `$TRESOS_BASE\plugins\<Crypto>\config_ext`. `$TRESOS_BASE` is the directory into which you have installed EB tresos Studio and `<Crypto>` is the specific `Crypto` module.

3.3.2.3. Initialization during startup

The configuration of a key element provides the parameter `CryptoKeyElementInitValue`. The value is used to initialize all instances of the key element in all configured keys during startup of the `Crypto` module. If no init value is configured, the default value of 0 is used if initialization is required. Further, the initialization sets the valid status of the configured keys. The initialization during startup of the `Crypto` module is done as follows:

- All non-persistent key elements are initialized.
- Persistent key elements are checked for NvM errors during NvM_ReadAll(). The Crypto module initializes a key element in the following cases:
  - if a key element could not be loaded successfully by the NvM, i.e. the result was not NVM_REQ_OK,
  - if the NvM has not yet initialized the key element, i.e. the result was not NVM_REQ_RESTORED_FROM_ROM.
- Typically, the key is set to valid. If at least one persistent key element of a key was initialized during startup of the Crypto module, the key is set to invalid.

For example, a block that was never written, is initialized during startup:

- by the NvM if so configured, e.g. with NvMRomBlockDataAccess. This does not result in an invalid key.
- for all other cases, by the Crypto with the configured initial value or the default value 0. The key is set to invalid.

After startup, the application may check the state of all keys. NvM_GetErrorStatus() returns the most recent error/status information, and thus also reveals e.g. NVM_REQ_RESTORED_FROM_ROM.

It is advised to check the state of the configured persistent keys after startup and to implement a handling suitable for the application if the key state is invalid.

NOTE **Identification of invalid keys**

The AUTOSAR specifications of the Crypto stack do not provide a function to retrieve a key's current valid state. This is probably introduced in revision 4.4 (see also RfC #79359). To identify invalid keys after startup, the application could e.g. perform service requests for all jobs that rely on a persistent key, and cancel these requests shortly afterward. If a key is invalid, the Crypto rejects the corresponding job with return value CRYPTO_E_KEY_NOT_VALID.

NOTE **Initialization and error recovery of key elements**

The NvM provides more sophisticated initialization and error recovery features, e.g. configuration of a redundant block, a user-defined callback function for initialization, or loading of a default value from ROM. The configuration of the init value of key elements should be integrated into an overall initialization and error recovery concept.

3.3.2.4. Key state

A key has a state. The cryptographic primitives only process a key if the key state is valid. The state of a key is set to valid by calling the Crypto_KeyValidSet() function. This function then stores all persistent key elements contained in the key to non-volatile memory by calling NvM_WriteBlock().

NOTE

Usage of the `Crypto_KeyValidSet()` call

The `Crypto_KeyValidSet()` function results in as many `NvM_WriteBlock()` calls as the given key contains persistent key elements. Therefore, it is recommended to call `Crypto_KeyValidSet()` only once, after all changes to contained key elements have been performed.

If an error occurs when calling `NvM_WriteBlock()`, the `Crypto` is not able to handle this error appropriately. `Crypto_KeyValidSet()` still sets the key state to valid and returns `E_NOT_OK` to indicate a problem due to `NvM` access.

NOTE

NvM error handling

The `Crypto` is not able to perform an error recovery for problems during storage of key elements to `NvM`. Therefore, it is recommended to check the return value of `Crypto_KeyValidSet()` at least for critical data. An example draft for handling the key's valid state could be: If `E_NOT_OK` is returned, call `Crypto_KeyValidSet()` again for a certain number of times. If `NvM` still could not write the block, call `NvM_GetErrrorStatus()` to identify the problematic key element. Then, call `NvM_WriteBlock()` for the corresponding block. Depending on the application's needs, it might implement a special error handling for the dedicated `NvM` block.

Further, as `Crypto_KeyValidSet()` is a synchronous function, the `Crypto` will not wait till the `NvM` has processed the write request, and will return to its callee directly. Thus, there will be no information available in the `Crypto` if the data has actually been stored to non-volatile memory successfully.

NOTE

NvM timing of storing data

According to the recommendation of the `NvM`, `SWS_NvM_00698`, after calling `NvM_WriteBlock()`, "the application must not modify the RAM block until success or failure of the request is signaled or derived via polling. In the meantime the contents of the RAM block may be read." Thus, if key element data is modified after calling `NvM_WriteBlock()`, e.g. by another key management function, but before the `NvM` has processed the data, the data stored to non-volatile memory might not be accurate.

3.3.2.5. Key element permissions

A key element has the configuration parameters `CryptoKeyElementReadAccess` and `CryptoKeyElementWriteAccess`. These parameters specify the permissions for accessing the key element. If `CryptoKeyElementReadAccess` of a key element is set to `CRYPTO_RA_INTERNAL_COPY`, it must be ensured that the key element cannot be read from outside the `Crypto`. To guarantee this, `Crypto_KeyElementCopy()` and `Crypto_KeyCopy()` also check the `CryptoKeyElementReadAccess` of the target key element.

In addition, if `CryptoKeyElementReadAccess` is set to `CRYPTO_RA_ALLOWED`, the key material can be read in plaintext, while `CRYPTO_RA_DENIED` blocks the read access from outside the `Crypto`. The same applies to the `CRYPTO_WA_*` constants.

NOTE

No general check of key element permissions

The AUTOSAR Specification of Crypto Driver [4] does not specify that the key management functions, except e.g. `Crypto_KeyElementGet/Set()`, shall check a key element's read and write access permissions. This is under discussion and might change in future AUTOSAR releases (see also RfC #80516). Currently, the APIs of the affected key management functions do not contain the appropriate return values `CRYPTO_E_KEY_READ/WRITE_FAIL`. Hence, the following key management functions do not check a key element's permissions: `Crypto_RandomSeed()`, `Crypto_KeyGenerate()`, `Crypto_KeyDerive()`, `Crypto_KeyExchangeCalcPubVal()`, `Crypto_KeyExchangeCalcSecret()`, `Crypto_CertificateParse()`, `Crypto_CertificateVerify()`.

3.3.3. Additional algorithm information

The following sections provide additional information about the use of specific algorithms.

3.3.3.1. Advanced Encryption Standard Electronic Code Book Mode (AES ECB)

The AES is a block cipher. So, only input sizes which are of one block size (16 bytes) are supported per UPDATE call. The result is written to the output buffer when the UPDATE is called.

3.3.3.2. Advanced Encryption Standard Cipher Block Chaining Mode (AES CBC)

The AES is a block cipher. So, only input sizes which are a multiple of the block size (16 bytes) are supported. If CBC is used, the input data is padded to fulfill this requirement. The padded bytes can only be removed if the complete input data is given to the decryption function. Streaming is supported by `Crypto`. Therefore, calling the CBC with mode FINISH indicates that the complete input data was provided. This can lead to unexpected behavior when calling CBC decrypt with mode UPDATE and input size 16 byte because the result length is zero in this case. The result is written to the output buffer when either UPDATE is called with the next block of input data or FINISH is called.

3.3.3.3. Advanced Encryption Standard Cipher Feedback Mode (AES CFB)

The CFB algorithm uses a block cipher (AES) for encryption and decryption. The underlying block cipher only supports keys with a length of 128 bits, 192 bits or 256 bits. Therefore, the CFB algorithm also only supports

keys with the same sizes. Calling the CFB algorithm with an unsupported key size fails as soon as the underlying block cipher is called.

The segment size of the CFB is set to 16 byte (CFB-128). The CFB is implemented without padding.

3.3.3.4. Advanced Encryption Standard Galois Counter Mode (AES GCM)

The GCM algorithm uses a block cipher (AES) to calculate the ciphertext and the tag. The underlying block cipher only supports keys with a length of 128 bits, 192 bits or 256 bits. Therefore, the GCM algorithm also only supports keys with the same sizes. Calling the GCM algorithm with an unsupported key size fails as soon as the underlying block cipher is called.

3.3.4. Certificate management

The certificate management functionality consists of the following two main service APIs:

- Certificate parse
- Certificate verify

3.3.4.1. Certificate parse

The certificate parse functionality takes a certificate stored in a key element `CRYPTO_KE_CERTIFICATE_DATA` of a certificate key and parses it. The format in which the certificate is stored is indicated in the key element `CRYPTO_KE_CERTIFICATE_PARSING_FORMAT`. Therefore, these key elements must be set before calling the certificate parse API. When successful, the certificate parse API extracts the following information from a certificate and stores it inside the corresponding `CryptoKeyElement`.

Certificate Data	CryptoKeyElement
Subject data	`CRYPTO_KE_CERTIFICATE_SUBJECT`
Signature	`CRYPTO_KE_CERTIFICATE_SIGNATURE`
Public key	`CRYPTO_KE_CERTIFICATE_SUBJECT_PUBLIC_KEY`
Signature algorithm	`CRYPTO_KE_CERTIFICATE_SIGNATURE_ALGORITHM`
Validity start date	`CRYPTO_KE_CERTIFICATE_VALIDITY_NOT_BEFORE`
Validity end (expiry) date	`CRYPTO_KE_CERTIFICATE_VALIDITY_NOT_AFTER`

3.3.4.2. Certificate verify

The certificate verify functionality verifies that the incoming certificate is issued by an authorized certificate issuer. For this, a public key is obtained from a root certificate that was obtained from the issuer authority. This public key verifies the signature of the incoming certificate. If the certificate is verified, the key that contains the incoming certificate is set to valid. If the incoming certificate is not verified, the key containing that certificate is set to invalid.

NOTE **Referencing the signature verification primitive**

The supported primitive, i.e. the cryptographic algorithm, for signature verification must be referenced in the Crypto Driver Object. This enables the primitive needed for certificate verification.

3.4. Job processing

The Crypto Driver Object can line up jobs into a queue to process them one after the other. The queue sorts the jobs according to the priority of the configured jobs. The higher the job priority value, the higher the job's priority.

Queuing can be disabled for a Crypto Driver Object by configuring the `CryptoQueueSize` parameter to `0`.

If only synchronous jobs are configured in the `Csm`, the scheduling of the `CryptoMainFunction` can be disabled via the `CryptoMainFunctionPeriod` parameter.

3.4.1. Exclusive areas

Each Crypto Driver Object can process one job at a time. Therefore, the internal state that marks the Crypto Driver Object as busy has to be secured with an exclusive area to prevent multiple jobs from being passed to the Crypto Driver Object. To prevent a task switch to another task that tries to pass the same `Csm` job to the Crypto Driver Object, the scheduling is disabled before reading the current state of the Crypto Driver Object and is reactivated after the Crypto Driver Object is marked as busy.

The same strategy is used for the queue within a Crypto Driver Object. As multiple tasks could attempt to enqueue a job into the queue, the queue operations (enqueue, dequeue, and remove) have to be secured with a global interrupt lock.

In some cases, it is required to call the exclusive area that secures the queue inside the exclusive area securing the Crypto Driver Object. This should be kept in mind when configuring the exclusive areas.

3.4.1.1. CRYPTO_EXCLUSIVE_AREA_DRIVEROBJECT

Protected data structures	The Crypto Driver Object that shall be protected from mutual access.
Recommended locking mechanism	This exclusive area must always be protected by a locking mechanism. The options for locking are described in the `EB tresos AutoCore Generic documentation`. Refer to the section `Mapping exclusive areas in the basic software modules` in the `Integration notes` section for details.

3.4.1.2. Crypto_SCHM_CRYPTO_EXCLUSIVE_AREA_QUEUE

Protected data structures	The Crypto queue that shall be protected from mutual access.
Recommended locking mechanism	This exclusive area must always be protected by a locking mechanism. The options for locking are described in the `EB tresos AutoCore Generic documentation`. Refer to the section `Mapping exclusive areas in the basic software modules` in the `Integration notes` section for details.

3.4.2. Channels with mixed job processing type

A channel is the path from a `Csm` queue via `CryIf` to a specific Crypto Driver Object in the `Crypto`. Channels may manage jobs of asynchronous and synchronous processing type. The use of mixed job processing types may lead to difficulties when activating a synchronous job.

The situation is illustrated in Figure 3.5, “Example situation for a mixed channel with both synchronous and asynchronous jobs”. Service calls for a synchronous job are executed at several points. As long as an asynchronous job is processed, the synchronous job is rejected. After job 1 is finished, there is a small time window until the next call of the main function, during which a synchronous service call can be accepted. In the next main function call, job 2 is dequeued from the queue and its processing starts.

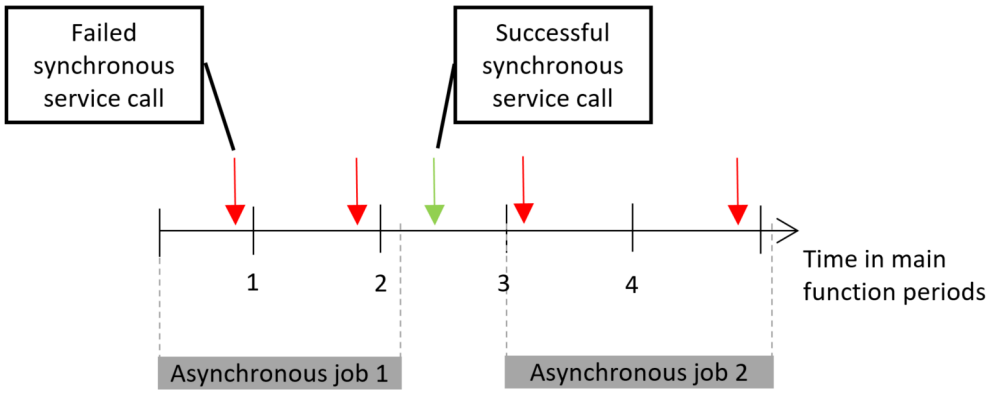

Figure 3.5. Example situation for a mixed channel with both synchronous and asynchronous jobs

The situation differs depending on the queue configuration. If a two-staged queuing is configured, the queue where the problem arises is the `Crypto` queue. If the `Crypto` has no queue configured, the `Csm` queue encounters the described problems. The problem occurs at the queue where the last dequeuing of a job is done.

With appropriate scheduling, you can avoid the described problems. The schedule for calling asynchronous jobs should include a time window for processing synchronous jobs.

3.4.3. Using the Crypto_ProcessJob() function

If `Crypto_ProcessJob()` writes to an output buffer (i.e. `outputPtr` or `secondaryOutputPtr`), it starts writing the computed value at position 0 of the passed output pointer. `Crypto_ProcessJob()` writes the actual length of the output buffer after the computation to the output length pointer (i.e. `outputLengthPtr` or `secondaryOutputLengthPtr`). The length is the overall length of the complete computation done in the current call of `Crypto_ProcessJob()`. Depending on the calling mode, the following values are written:

- If `Crypto_ProcessJob()` is called with mode CRYPTO_OPERATIONMODE_UPDATE, it writes the length of the output buffer that results from the UPDATE operation only.
- If `Crypto_ProcessJob()` is called with mode CRYPTO_OPERATIONMODE_FINISH, it writes the length of the output buffer that results from the FINISH operation only.
- If `Crypto_ProcessJob()` is called with mode CRYPTO_OPERATIONMODE_SINGLECALL, it writes the length of the output buffer of UPDATE and FINISH operations.

NOTE **Handling of the output pointer**

`Crypto_ProcessJob()` starts writing to the output buffer at position 0 for every call. Therefore, it might be necessary to update the pointer to the output buffer before a new call to `Crypto_ProcessJob()` or store the result in such a way that it can be overwritten.

3.5. Configuring the Crypto module

This chapter provides you with information on how to configure the `Crypto` module in EB tresos Studio. To understand how to configure the `Crypto` module, you must be familiar with the basic concepts of the `Crypto`. For information on the `Crypto` concepts, see Section 3.2, “Background information”.

NOTE **Dependency of `Csm`, `CryIf`, and `Crypto`**

To perform a certain cryptographic operation, you must configure `Csm`, `CryIf`, and `Crypto`. For the configuration of `Csm` and `CryIf`, see the ACG8 Crypto and Security Stack documentation.

3.5.1. Configuring a Crypto key

With the **Crypto_preconf** preconfiguration of the `Crypto`, the following objects are already configured and allow only limited editing:

- **Cryptographic primitives:** Open the `CryptoPrimitives` tab to see all primitives that the `Crypto` supports.
- **Crypto Driver Object:** Open the `CryptoDriverObjects` tab. Double-click an entry to view all referenced `CryptoPrimitives`. If desired, a Crypto Driver Object can provide its own queue to line up asynchronous jobs. To use queuing, open the Crypto Driver Object and set the `CryptoQueueSize` parameter to a value higher than `0`.

The preconfiguration does not cover the configuration of keys. For each key that shall be used within the `Crypto`, you must create a separate entry.

Configuring the Crypto

Prerequisite:

- The **Crypto_preconf** is applied. For information on how to add a preconfiguration, see the EB tresos Studio documentation.

- NvRam blocks are configured for persistent Crypto key elements. For information on how to configure NvRam blocks, see Section 3.5.2, “Configuring persistent storing of key elements”.

Step 1
On the **CryptoKeys** tab, add a new entry.

Step 2
In the `CryptoKeyTypeRef` parameter, select a Crypto key type. To view the key elements of a key type, click the **CryptoKeyTypes** tab and open the desired key type.

Step 3
To save a Crypto key persistently:

Step 3.1
In the `Crypto/CryptoKey/CryptoKeyNvRamBlockIds` parameter, create an `NvRamBlockID` that references the specific key element.

Step 3.2
Select the corresponding `NVRAM Block`. Each key that contains a persistent key element needs an individual NvRam block.

You can also create new `CryptoKeyType` entries, accustomed to your needs. Consider that the provided cryptographic primitives expect a key type that contains certain key elements as specified in SWS_Csm_01022 of the AUTOSAR Specification of Crypto Service Manager [1].

3.5.2. Configuring persistent storing of key elements

The `Crypto` allows to configure key elements to be stored persistently, using the parameter `CryptoKeyElementPersist`. This requires the configuration of objects in the `NvM` module. For each persistent key element of a key, an appropriate NvRam block is needed.

Configuring the `NvM`

Step 1
Configure the `NvMSelectBlockForReadAll` parameter to `TRUE`.

Step 2
Configure the `NvMSelectBlockForWriteAll` parameter to `TRUE`.

Step 3
Configure the `NvMNvBlockLength` parameter equal to the size of the Crypto data structure of the corresponding key element. To determine the length of each NvRam block used for a `Crypto` key element:

Step 3.1
Configure the `NvMExtraBlockChecks` parameter to `FALSE` to disable extra block checks for the NvRam block. Otherwise, the `NvM` checks the block size during compilation time and might not compile.

Step 3.2
Compile the project.

Step 3.3
Get the size of NvRam block, e.g. from the map file.

Step 3.4
Configure the block length to this NvRam block size.

Step 3.5
Re-enable extra block checks if needed.

Step 4
Optionally, enable CRC. This is recommended but not mandatory.

NOTE

Improving the persistent storage of key elements

Persistent key elements are written to non-volatile memory by calling `NvM_WriteBlock()` when the corresponding key is set to valid, i.e. when `Crypto_KeyValidSet()` is called. You can optimize the storage of key elements to `NvM` through further configuration in the `NvM`. The `NvM` configuration provides the parameters `NvMPreWriteDataComp` and `NvMBlockUseCRCCompMechanism`. You can set these parameters so that the `NvM` only writes to memory if key elements were actually modified.

NOTE

Controlling stored key elements

In order to gain full control over the storage of your key elements, you might consider to not define a key element as persistent. Instead, you might manage an `NvM` block from your application, read the content of the key element at dedicated points, and write to non-volatile memory only as required.

WARNING

Security implications for the use of `NvM`

The `Crypto` allows to use the `NvM` to store key elements persistently. The `Crypto` cannot ensure that security conditions are maintained when using this feature. Consider additional measures to avoid security leaks.

3.5.3. Optimizing structure size

NOTE

Optimizing structure size

For following structures in `Crypto`, it is possible to define the length of the data array via optional Csm configuration parameters. Following dependencies apply:

- `Crypto_xVIx_xAIx_SymKeyType` can be adapted with the configuration parameter `CsmSymKeyMaxLength`
- `Crypto_xVIx_xAIx_AsymPrivateKeyType` can be adapted with the configuration parameter `CsmAsymPrivateKeyMaxLength`
- `Crypto_xVIx_xAIx_AsymPublicKeyType` can be adapted with the configuration parameter `CsmAsymPublicKeyMaxLength`

WARNING

Optimizing structure size risks

The integrator has to make sure, that the CsmAsymPrivateKeyMaxLength, CsmAsymPublicKeyMaxLength, and CsmSymKeyMaxLength configured in the `Csm` have at least the size, which is used for symmetric, public or private key elements in the `Crypto`, else the buffers are too small and it can lead to a buffer overflow.

4. Crypto module references

4.1. Configuration parameters

Containers included		
Container name	**Multiplicity**	**Description**
CommonPublishedInformation	1..1	**Label:** Common Published Information Common container, aggregated by all modules. It contains published information about vendor and versions.
CryptoDefensiveProgramming	1..1	**Label:** Defensive Programming Options Parameters for defensive programming
CryptoGeneral	1..1	**Label:** CryptoGeneral Container for common configuration options.
CryptoDriverObjects	1..1	**Label:** CryptoDriverObjects Container for CRYPTO Objects.
CryptoKeyElements	0..1	**Label:** CryptoKeyElements Container for Crypto key elements.
CryptoKeyTypes	0..1	**Label:** CryptoKeyTypes Container for CRYPTO key types.
CryptoKeys	0..1	**Label:** CryptoKeys Container for CRYPTO keys.
CryptoPrimitives	0..n	**Label:** CryptoPrimitives Container for CRYPTO primitives.
PublishedInformation	1..1	**Label:** EB Published Information Additional published parameters not covered by CommonPublishedInformation container.

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT	
Label	Config Variant	
Description	Select the configuration variant. Currently only PreCompile is supported.	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	VariantPreCompile	
Range	VariantPreCompile	
Configuration class	**VariantPreCompile:**	VariantPreCompile

4.1.1. CommonPublishedInformation

Parameters included	
Parameter name	**Multiplicity**
ArMajorVersion	1..1
ArMinorVersion	1..1
ArPatchVersion	1..1
SwMajorVersion	1..1
SwMinorVersion	1..1
SwPatchVersion	1..1
ModuleId	1..1
VendorId	1..1
Release	1..1
VendorApiInfix	1..1

Parameter Name	ArMajorVersion
Label	AUTOSAR Major Version
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL

Default value	4
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**ArMinorVersion**
Label	AUTOSAR Minor Version
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	3
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**ArPatchVersion**
Label	AUTOSAR Patch Version
Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	0
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**SwMajorVersion**
Label	Software Major Version
Description	Major version number of the vendor specific implementation of the module.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	1
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**SwMinorVersion**

Label	Software Minor Version
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	7
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**SwPatchVersion**
Label	Software Patch Version
Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	26
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**ModuleId**
Label	Numeric Module ID
Description	Module ID of this module from Module List
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	114
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**VendorId**
Label	Vendor ID
Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
Multiplicity	1..1
Type	INTEGER_LABEL

Default value	1
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**Release**
Label	Release Information
Multiplicity	1..1
Type	STRING_LABEL
Default value	
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

Parameter Name	**VendorApiInfix**
Label	VendorApiInfix
Description	Vendor Api Infix of the dedicated implementation of this module according to the AUTOSAR
Multiplicity	1..1
Type	STRING
Origin	Elektrobit Automotive GmbH

4.1.2. CryptoDefensiveProgramming

Parameters included	
Parameter name	**Multiplicity**
CryptoDefProgEnabled	1..1
CryptoPrecondAssertEnabled	1..1
CryptoPostcondAssertEnabled	1..1
CryptoStaticAssertEnabled	1..1
CryptoUnreachAssertEnabled	1..1
CryptoInvariantAssertEnabled	1..1

Parameter Name	**CryptoDefProgEnabled**
Label	Enable Defensive Programming
Description	Enables or disables the defensive programming feature for the module Crypto.

	Note: This feature is dependent on the use of the development error detection module. To use the defensive programming feature, proceed as follows: 1. Enable development error detection 2. Enable defensive programming 3. Enable assertions as required
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**CryptoPrecondAssertEnabled**
Label	Enable Precondition Assertions
Description	Enables handling of precondition assertion checks reported from the module Crypto. Dependency on parameter(s): ► Enable Development Error Detection (`CryptoDevErrorDetect`): must be enabled ► Enable Defensive Programming (`CryptoDefProgEnabled`): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**CryptoPostcondAssertEnabled**
Label	Enable Postcondition Assertions
Description	Enables handling of postcondition assertion checks reported from the module Crypto. Dependency on parameter(s): ► Enable Development Error Detection (`CryptoDevErrorDetect`): must be enabled

	► Enable Defensive Programming (`CryptoDefProgEnabled`): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**CryptoStaticAssertEnabled**
Label	Enable Static Assertions
Description	Enables handling of static assertion checks reported from the module Crypto. Dependency on parameter(s): ► Enable Development Error Detection (`CryptoDevErrorDetect`): must be enabled ► Enable Defensive Programming (`CryptoDefProgEnabled`): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	**CryptoUnreachAssertEnabled**
Label	Enable Unreachable Code Assertions
Description	Enables handling of unreachable code assertion checks reported from the module Crypto. Dependency on parameter(s): ► Enable Development Error Detection (`CryptoDevErrorDetect`): must be enabled ► Enable Defensive Programming (`CryptoDefProgEnabled`): must be enabled
Multiplicity	1..1
Type	BOOLEAN

Default value	false	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	**CryptoInvariantAssertEnabled**	
Label	Enable Invariant Assertions	
Description	Enables handling of invariant assertion checks reported from functions of the module Crypto. Dependency on parameter(s): ► Enable Development Error Detection (`CryptoDevErrorDetect`): must be enabled ► Enable Defensive Programming (`CryptoDefProgEnabled`): must be enabled	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

4.1.3. CryptoGeneral

Parameters included	
Parameter name	**Multiplicity**
CryptoDevErrorDetect	1..1
CryptoInstanceId	1..1
CryptoMainFunctionPeriod	0..1
CryptoVersionInfoApi	1..1

Parameter Name	**CryptoDevErrorDetect**
Label	CryptoDevErrorDetect
Description	Switches the development error detection and notification on or off. TRUE = detection and notification is enabled.

	FALSE = detection and notification is disabled.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	False	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoInstanceId**	
Label	CryptoInstanceId	
Description	Instance ID of the crypto driver. This ID is used to discern several crypto drivers in case more than one driver is used in the same ECU.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoMainFunctionPeriod**	
Label	CryptoMainFunctionPeriod	
Description	Specifies the period of main function Crypto_MainFunction in seconds.	
Multiplicity	0..1	
Type	FLOAT	
Default value	0.01	
Range	>0	
Configuration class	**PreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoVersionInfoApi**
Label	CryptoVersionInfoApi
Description	Pre-processor switch to enable and disable availability of the API Crypto_GetVersionInfo(). TRUE = API Crypto_GetVersionInfo() is available. FALSE = API Crypto_GetVersionInfo() is not available.

Multiplicity	1..1	
Type	BOOLEAN	
Default value	False	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

4.1.4. CryptoDriverObjects

Containers included		
Container name	**Multiplicity**	**Description**
CryptoDriverObject	0..n	**Label:** CryptoDriverObject Configuration of a CryptoDriverObject.

4.1.5. CryptoDriverObject

Parameters included	
Parameter name	**Multiplicity**
CryptoDriverObjectId	1..1
CryptoPrimitiveRef	0..n
CryptoQueueSize	1..1

Parameter Name	**CryptoDriverObjectId**	
Label	CryptoDriverObjectId	
Description	Identifier of the Crypto Driver Object. The Crypto Driver Object offers different crypto primitives.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	>=0 <=4294967295	
Configuration class	**VariantPreCompile:**	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	**CryptoPrimitiveRef**	
Label	CryptoPrimitiveRef	
Description	Refers to primitive in the CRYPTO. The CryptoPrimitive is a pre-configured container of the crypto service that shall be used.	
Multiplicity	0..n	
Type	REFERENCE	
Range	node:paths(./../../../../../CryptoPrimitives/*/CryptoPrimitive/*)	
Configuration class	**PreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoQueueSize**	
Label	CryptoQueueSize	
Description	Size of the queue in the Crypto Driver. Defines the maximum number of jobs in the Crypto Driver Object queue. If it is set to 0, queueing is disabled in the Crypto Driver Object.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	>=0 <=4294967295	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

4.1.6. CryptoKeyElements

Containers included		
Container name	**Multiplicity**	**Description**
CryptoKeyElement	1..n	**Label:** CryptoKeyElement Configuration of a CryptoKeyElement.

4.1.7. CryptoKeyElement

Parameters included	
Parameter name	**Multiplicity**
CryptoKeyElementAllowPartialAccess	1..1
CryptoKeyElementId	1..1
CryptoKeyElementInitValue	1..1
CryptoKeyElementPersist	1..1
CryptoKeyElementReadAccess	1..1
CryptoKeyElementSize	1..1
CryptoKeyElementWriteAccess	1..1

Parameter Name	**CryptoKeyElementAllowPartialAccess**	
Label	CryptoKeyElementAllowPartialAccess	
Description	Enable or disable writing and reading the key element with data smaller than the size of the element. TRUE = enable partial access of the key element. FALSE = disable partial access of the key element.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	False	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoKeyElementId**	
Label	CryptoKeyElementId	
Description	Identifier of the CRYPTO key element.	
Multiplicity	1..1	
Type	INTEGER	
Default value	1	
Range	>=0 --- <=4294967295	
Configuration class	**VariantPreCompile:**	VariantPreCompile

Origin	AUTOSAR_ECUC

Parameter Name	**CryptoKeyElementInitValue**	
Label	CryptoKeyElementInitValue	
Description	Value which will be used to fill the element during initialization, when the element is not already initialized. The value is parsed as comma-separated byte values given in hexadecimal representation (uint8 array). E.g. 0x12, 0xab, 0xff would be a valid input. If initialization value contains less hexadecimal values than configured by CryptoKeyElementSize the array is filled with 0x00 in the end.	
Multiplicity	1..1	
Type	STRING	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoKeyElementPersist**	
Label	CryptoKeyElementPersist	
Description	Enable or disable persisting of the key element in non-volatile storage. TRUE = enable persisting of the key element. FALSE = disable persisting of the key element.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	False	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoKeyElementReadAccess**
Label	CryptoKeyElementReadAccess
Description	Define the reading access rights of the key element. CRYPTO_RA_DENIED = key element cannot be read from outside the Crypto Driver. CRYPTO_RA_INTERNAL_COPY = key element can be copied to another key element in the same Crypto Driver. CRYPTO_RA_ALLOWED = key element can be read as plaintext.
Multiplicity	1..1

Type	ENUMERATION
Default value	CRYPTO_RA_DENIED
Range	CRYPTO_RA_ALLOWED CRYPTO_RA_DENIED CRYPTO_RA_INTERNAL_COPY
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**CryptoKeyElementSize**
Label	CryptoKeyElementSize
Description	Maximum size of a Crypto key element in bytes.
Multiplicity	1..1
Type	INTEGER
Default value	1
Range	>=1 <=4294967295
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**CryptoKeyElementWriteAccess**
Label	CryptoKeyElementWriteAccess
Description	Define the writing access rights of the key element. CRYPTO_WA_DENIED = key element can not be written from outside the Crypto Driver. CRYPTO_WA_INTERNAL_COPY = key element can be filled with another key element in the same Crypto Driver. CRYPTO_WA_ALLOWED = key element can be written as plaintext.
Multiplicity	1..1
Type	ENUMERATION
Default value	CRYPTO_WA_DENIED
Range	CRYPTO_WA_ALLOWED CRYPTO_WA_DENIED CRYPTO_WA_INTERNAL_COPY

Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

4.1.8. CryptoKeyTypes

Containers included		
Container name	**Multiplicity**	**Description**
CryptoKeyType	1..n	**Label:** CryptoKeyType Configuration of a CryptoKeyType.

4.1.9. CryptoKeyType

Parameters included	
Parameter name	**Multiplicity**
CryptoKeyElementRef	1..n

Parameter Name	**CryptoKeyElementRef**	
Label	CryptoKeyElementRef	
Description	Refers to a CryptoKeyElement, which holds the configuration of the crypto key element.	
Multiplicity	1..n	
Type	REFERENCE	
Range	node:paths(./../../../../../CryptoKeyElements/CryptoKeyElement/*)	
Configuration class	**PreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

4.1.10. CryptoKeys

Containers included		
Container name	**Multiplicity**	**Description**
CryptoKey	1..n	**Label:** CryptoKey

Containers included		
		Configuration of a CryptoKey.

4.1.11. CryptoKey

Containers included		
Container name	**Multiplicity**	**Description**
CryptoKeyNvRamBlockIds	0..n	**Crypto Block Configuration** This container contains the configuration (parameters) for a non-volatile memory block reference, which is used from the Crypto. If the permanent storage of a key element entry is required, a valid NvM block reference needs to be configured. Otherwise no NvM block and block reference need to be configured, so the data is stored only volatile. The required NvM block references (CryptoNvRamBlockIds) need to be configured by the user. The user needs to create and configure the NvM blocks for Crypto. **Integration Note:** Do NOT trigger external write requests for any Crypto designated non-volatile memory block as it is possible for this external trigger to block eventual internal write processing in the Crypto, causing DET warnings to be reported and the internal request to stop as the NvM will be busy handling the external requests for the specific memory block.

Parameters included	
Parameter name	**Multiplicity**
CryptoKeyDeriveIterations	1..1
CryptoKeyId	1..1
CryptoKeyTypeRef	1..1

Parameter Name	CryptoKeyDeriveIterations
Label	CryptoKeyDeriveIterations
Description	Holds the number of iterations to be performed by the key derivation primitive.
Multiplicity	1..1

Type	INTEGER	
Default value	1	
Range	>=1 <=4294967295	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoKeyId**
Label	CryptoKeyId
Description	Identifier of the CRYPTO Key.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	>=0 <=4294967295
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	**CryptoKeyTypeRef**
Label	CryptoKeyTypeRef
Description	Refers to a CryptoKeyType. The CryptoKeyType provides the information which key elements are contained in a CryptoKey.
Multiplicity	1..1
Type	REFERENCE
Range	node:paths(./../../../../CryptoKeyTypes/CryptoKeyType/*)
Configuration class	**VariantPreCompile:** VariantPreCompile
Origin	AUTOSAR_ECUC

4.1.12. CryptoKeyNvRamBlockIds

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
CryptoPersistKeyElement	1..1
CryptoNvramBlockIdRef	1..1

Parameter Name	CryptoPersistKeyElement	
Label	CryptoPersistKeyElement	
Description	This parameter holds the persistent key element.	
Multiplicity	1..1	
Type	SYMBOLIC-NAME-REFERENCE	
Range	node:paths(./../../../../../../CryptoKeyElements/CryptoKeyElement/*)	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC V1.0.0	

Parameter Name	CryptoNvramBlockIdRef	
Label	CryptoNvramBlockIdRef	
Description	This parameter references to the NvmBlockDescriptor for NVRAM Blocks.	
Multiplicity	1..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC V1.0.0	

4.1.13. CryptoPrimitives

Containers included		
Container name	**Multiplicity**	**Description**
CryptoPrimitive	0..n	**Label:** CryptoPrimitive Configuration of a CryptoPrimitive.

4.1.14. CryptoPrimitive

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
CryptoPrimitiveAlgorithmFamiliy	1..n
CryptoPrimitiveAlgorithmMode	1..n
CryptoPrimitiveAlgorithmSecondaryFamiliy	1..n
CryptoPrimitiveService	1..1

Parameter Name	CryptoPrimitiveAlgorithmFamiliy
Label	CryptoPrimitiveAlgorithmFamiliy
Description	Determines the algorithm family used for the crypto service.
Multiplicity	1..n
Type	ENUMERATION
Default value	CRYPTO_ALGOFAM_3DES
Range	CRYPTO_ALGOFAM_3DES
	CRYPTO_ALGOFAM_AES
	CRYPTO_ALGOFAM_BLAKE_1_256
	CRYPTO_ALGOFAM_BLAKE_1_512
	CRYPTO_ALGOFAM_BLAKE_2s_256
	CRYPTO_ALGOFAM_BLAKE_2s_512
	CRYPTO_ALGOFAM_BRAINPOOL
	CRYPTO_ALGOFAM_CHACHA
	CRYPTO_ALGOFAM_CUSTOM
	CRYPTO_ALGOFAM_ECCNIST
	CRYPTO_ALGOFAM_ECIES
	CRYPTO_ALGOFAM_ED25519
	CRYPTO_ALGOFAM_NOT_SET
	CRYPTO_ALGOFAM_RIPEMD160
	CRYPTO_ALGOFAM_RNG
	CRYPTO_ALGOFAM_RSA
	CRYPTO_ALGOFAM_SECURECOUNTER
	CRYPTO_ALGOFAM_SHA1
	CRYPTO_ALGOFAM_SHA2_224
	CRYPTO_ALGOFAM_SHA2_256
	CRYPTO_ALGOFAM_SHA2_384

	CRYPTO_ALGOFAM_SHA2_512	
	CRYPTO_ALGOFAM_SHA2_512_224	
	CRYPTO_ALGOFAM_SHA2_512_256	
	CRYPTO_ALGOFAM_SHA3_224	
	CRYPTO_ALGOFAM_SHA3_256	
	CRYPTO_ALGOFAM_SHA3_384	
	CRYPTO_ALGOFAM_SHA3_512	
	CRYPTO_ALGOFAM_SHAKE128	
	CRYPTO_ALGOFAM_SHAKE256	
	CRYPTO_ALGOFAM_SIPHASH	
Configuration class	**PreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoPrimitiveAlgorithmMode**
Label	CryptoPrimitiveAlgorithmMode
Description	Determines the algorithm mode used for the crypto service.
Multiplicity	1..n
Type	ENUMERATION
Default value	CRYPTO_ALGOMODE_12ROUNDS
Range	CRYPTO_ALGOMODE_12ROUNDS
	CRYPTO_ALGOMODE_20ROUNDS
	CRYPTO_ALGOMODE_8ROUNDS
	CRYPTO_ALGOMODE_CBC
	CRYPTO_ALGOMODE_CFB
	CRYPTO_ALGOMODE_CMAC
	CRYPTO_ALGOMODE_CTR
	CRYPTO_ALGOMODE_CTRDRBG
	CRYPTO_ALGOMODE_CUSTOM
	CRYPTO_ALGOMODE_ECB
	CRYPTO_ALGOMODE_GCM
	CRYPTO_ALGOMODE_GMAC
	CRYPTO_ALGOMODE_HMAC
	CRYPTO_ALGOMODE_NOT_SET

	CRYPTO_ALGOMODE_OFB	
	CRYPTO_ALGOMODE_RSAES_OAEP	
	CRYPTO_ALGOMODE_RSAES_PKCS1_v1_5	
	CRYPTO_ALGOMODE_RSASSA_PKCS1_v1_5	
	CRYPTO_ALGOMODE_RSASSA_PSS	
	CRYPTO_ALGOMODE_XTS	
	CRYPTO_ALGOMODE_SIPHASH_2_4	
	CRYPTO_ALGOMODE_SIPHASH_4_8	
Configuration class	**PreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoPrimitiveAlgorithmSecondaryFamiliy**
Label	CryptoPrimitiveAlgorithmSecondaryFamiliy
Description	Determines the algorithm secondary family used for the crypto service.
Multiplicity	1..n
Type	ENUMERATION
Default value	CRYPTO_ALGOFAM_NOT_SET
Range	CRYPTO_ALGOFAM_3DES
	CRYPTO_ALGOFAM_AES
	CRYPTO_ALGOFAM_BLAKE_1_256
	CRYPTO_ALGOFAM_BLAKE_1_512
	CRYPTO_ALGOFAM_BLAKE_2s_256
	CRYPTO_ALGOFAM_BLAKE_2s_512
	CRYPTO_ALGOFAM_BRAINPOOL
	CRYPTO_ALGOFAM_CHACHA
	CRYPTO_ALGOFAM_CUSTOM
	CRYPTO_ALGOFAM_ECCNIST
	CRYPTO_ALGOFAM_ECIES
	CRYPTO_ALGOFAM_ED25519
	CRYPTO_ALGOFAM_NOT_SET
	CRYPTO_ALGOFAM_RIPEMD160
	CRYPTO_ALGOFAM_RNG
	CRYPTO_ALGOFAM_RSA

	CRYPTO_ALGOFAM_SECURECOUNTER	
	CRYPTO_ALGOFAM_SHA1	
	CRYPTO_ALGOFAM_SHA2_224	
	CRYPTO_ALGOFAM_SHA2_256	
	CRYPTO_ALGOFAM_SHA2_384	
	CRYPTO_ALGOFAM_SHA2_512	
	CRYPTO_ALGOFAM_SHA2_512_224	
	CRYPTO_ALGOFAM_SHA2_512_256	
	CRYPTO_ALGOFAM_SHA3_224	
	CRYPTO_ALGOFAM_SHA3_256	
	CRYPTO_ALGOFAM_SHA3_384	
	CRYPTO_ALGOFAM_SHA3_512	
	CRYPTO_ALGOFAM_SHAKE128	
	CRYPTO_ALGOFAM_SHAKE256	
	CRYPTO_ALGOFAM_SIPHASH	
Configuration class	**PreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	**CryptoPrimitiveService**
Label	CryptoPrimitiveService
Description	Determines the crypto service used for defining the capabilities.
Multiplicity	1..1
Type	ENUMERATION
Default value	AEAD_DECRYPT
Range	AEAD_DECRYPT
	AEAD_ENCRYPT
	DECRYPT
	ENCRYPT
	HASH
	MAC_GENERATE
	MAC_VERIFY
	RANDOM
	SIGNATURE_GENERATE

	SIGNATURE_VERIFY	
Configuration class	**VariantPreCompile:**	VariantPreCompile
Origin	AUTOSAR_ECUC	

4.1.15. PublishedInformation

Parameters included	
Parameter name	**Multiplicity**
PbcfgMSupport	1..1

Parameter Name	**PbcfgMSupport**
Label	PbcfgM support
Description	Specifies whether or not the Crypto can use the PbcfgM module for post-build support.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	**PublishedInformation:**
Origin	Elektrobit Automotive GmbH

4.2. Application programming interface (API)

4.2.1. Type definitions

4.2.1.1. Crypto[_<Vi>_<Ai>]_CancelFuncPtr_t

Purpose	Pointer to a cancel function defined in the process layer.
Type	`Std_ReturnType(*)(uint32 objectId, Crypto_JobType *job)`

4.2.1.2. Crypto[_<Vi>_<Ai>]_DriverObject

Purpose	Structure of a crypto driver object.	
Type	struct	
Members	`boolean SkipPeriodicMainFunction`	
	`Crypto[_<Vi>_<Ai>]_DriverObjectStateType DriverObjectState`	
	`Crypto_JobType * CurrentJob`	
	`Crypto[_<Vi>_<Ai>]_QueueType *const Queue`	

4.2.1.3. Crypto[_<Vi>_<Ai>]_DriverObjectStateType

Purpose	Different states of the crypto driver object.
Type	`uint8`

4.2.1.4. Crypto[_<Vi>_<Ai>]_Key

Purpose	Structure of a crypto key.	
Type	struct	
Members	`const uint32 KeyDeriveIterations`	
	`const uint32 KeyElements`	
	`const Crypto[_<Vi>_<Ai>]_KeyElementPtr *const KeyType`	
	`Crypto[_<Vi>_<Ai>]_KeyStateType KeyState`	

4.2.1.5. Crypto[_<Vi>_<Ai>]_KeyElement

Purpose	Structure of a crypto key element.	
Type	struct	
Members	`uint32 Id`	

	boolean AllowPartialAccess	
	const uint8 *const InitValue	
	boolean Persist	
	const Crypto[_<Vi>_<Ai>]_- ReadAccessType ReadAccess	
	uint32 Size	
	const Crypto[_<Vi>_<Ai>]_- WriteAccessType WriteAccess	

4.2.1.6. Crypto[_<Vi>_<Ai>]_KeyElementPtr

Purpose	Pointer to a crypto key element.
Type	Crypto[_<Vi>_<Ai>]_KeyElement *

4.2.1.7. Crypto[_<Vi>_<Ai>]_KeyStateType

Purpose	Different states of a crypto key.
Type	boolean

4.2.1.8. Crypto[_<Vi>_<Ai>]_ProcessFuncPtr_t

Purpose	Pointer to a process function defined in the process layer.
Type	Std_ReturnType(*)(uint32 objectId, Crypto_JobType *job)

4.2.1.9. Crypto[_<Vi>_<Ai>]_QueueElementPtr

Purpose	Pointer to a crypto key element.
Type	Crypto[_<Vi>_<Ai>]_QueueElementType *

4.2.1.10. Crypto[_<Vi>_<Ai>]_QueueElementType

Purpose	Structure of a queue element.
Type	struct

Members	`Crypto_JobType * Job`	
	`struct Crypto[_<Vi>_<Ai>]_QueueElementType * Next`	
	`Crypto[_<Vi>_<Ai>]_ProcessFuncPtr_t ProcessFunction`	

4.2.1.11. Crypto[_<Vi>_<Ai>]_QueueType

Purpose	Meta data structure of a queue.	
Type	struct	
Members	`uint32 CurrentSize`	
	`const uint32 MaxSize`	
	`Crypto[_<Vi>_<Ai>]_QueueElementType * Head`	
	`Crypto[_<Vi>_<Ai>]_QueueElementType *const Data`	

4.2.1.12. Crypto[_<Vi>_<Ai>]_ReadAccessType

Purpose	Different read access types for a key element.
Type	`uint8`

4.2.1.13. Crypto[_<Vi>_<Ai>]_WriteAccessType

Purpose	Different write access types for a key element.
Type	`uint8`

4.2.2. Macro constants

4.2.2.1. CRYPTO[_<VI>_<AI>]_DATE_SIZE

Purpose	The size of a date in certificate.

Value	6U

4.2.2.2. CRYPTO[_<VI>_<AI>]_DRIVER_OBJECT_STATE_BUSY

Purpose	Crypto driver object is busy.
Value	0x01U

4.2.2.3. CRYPTO[_<VI>_<AI>]_DRIVER_OBJECT_STATE_IDLE

Purpose	Crypto driver object is idle.
Value	0x00U

4.2.2.4. CRYPTO[_<VI>_<AI>]_E_INIT_FAILED

Purpose	Initialization of crypto driver failed.
Value	0x01U

4.2.2.5. CRYPTO[_<VI>_<AI>]_E_PARAM_HANDLE

Purpose	Handle parameter has an invalid value.
Value	0x04U

4.2.2.6. CRYPTO[_<VI>_<AI>]_E_PARAM_POINTER

Purpose	Pointer parameter has an invalid value.
Value	0x02U

4.2.2.7. CRYPTO[_<VI>_<AI>]_E_PARAM_VALUE

Purpose	Value parameter has an invalid value.
Value	0x05U

4.2.2.8. CRYPTO[_<VI>_<AI>]_E_RE_ENTROPY_EXHAUSTED

Purpose	
Value	0x03U

4.2.2.9. CRYPTO[_<VI>_<AI>]_E_RE_KEY_NOT_AVAILABLE

Purpose	
Value	0x01U

4.2.2.10. CRYPTO[_<VI>_<AI>]_E_RE_KEY_READ_FAIL

Purpose	
Value	0x02U

4.2.2.11. CRYPTO[_<VI>_<AI>]_E_RE_SMALL_BUFFER

Purpose	
Value	0x00U

4.2.2.12. CRYPTO[_<VI>_<AI>]_E_UNINIT

Purpose	Crypto driver is not initialized.
Value	0x00U

4.2.2.13. CRYPTO[_<VI>_<AI>]_KEY_STATE_INVALID

Purpose	Key is invalid.
Value	0x00U

4.2.2.14. CRYPTO[_<VI>_<AI>]_KEY_STATE_VALID

Purpose	Key is valid.

Value	0x01U

4.2.2.15. CRYPTO[_<VI>_<AI>]_KE_AES_EXPANDEDKEY

Purpose	Key element which can be used together with the key element CRYPTO_KE_MAC_-KEY to precalculate the expanded AES key.
Value	1000U

4.2.2.16. CRYPTO[_<VI>_<AI>]_KE_MAC_AESCMAC_SUBKEY1

Purpose	Key element which can be used together with the key element CRYPTO_KE_MAC_-KEY to precalculate the AES-CMAC subkey K1.
Value	1001U

4.2.2.17. CRYPTO[_<VI>_<AI>]_KE_MAC_AESCMAC_SUBKEY2

Purpose	Key element which can be used together with the key element CRYPTO_KE_MAC_-KEY to precalculate the AES-CMAC subkey K2.
Value	1002U

4.2.2.18. CRYPTO[_<VI>_<AI>]_KE_RSA_ADDITIONAL_INPUT

Purpose	Key element which can be used to store the RSA additional input.
Value	1004U

4.2.2.19. CRYPTO[_<VI>_<AI>]_KE_SIGNATURE_BARRETT

Purpose	Key element which can be used to store the barrett of an RSA key.
Value	1003U

4.2.2.20. CRYPTO[_<VI>_<AI>]_RA_ALLOWED

Purpose	Key element can be read in plaintext.

Value	0x03U

4.2.2.21. CRYPTO[_<VI>_<AI>]_RA_DENIED

Purpose	The key element can not be read from outside the crypto driver.
Value	0x01U

4.2.2.22. CRYPTO[_<VI>_<AI>]_RA_ENCRYPTED

Purpose	Key element can be read encrypted.
Value	0x04U

4.2.2.23. CRYPTO[_<VI>_<AI>]_RA_INTERNAL_COPY

Purpose	Key element can be copied within the same crypto driver.
Value	0x02U

4.2.2.24. CRYPTO[_<VI>_<AI>]_SID_CANCELJOB

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_CancelJob.
Value	0x0EU

4.2.2.25. CRYPTO[_<VI>_<AI>]_SID_CERTIFICATEPARSE

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_CertificateParse.
Value	0x0BU

4.2.2.26. CRYPTO[_<VI>_<AI>]_SID_CERTIFICATEVERIFY

Purpose	AUTOSAR API service ID for Crypto_CertificateVerifiy.
Value	0x12U

4.2.2.27. CRYPTO[_<VI>_<AI>]_SID_GETVERSIONINFO

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_GetVersionInfo.
Value	0x01U

4.2.2.28. CRYPTO[_<VI>_<AI>]_SID_INIT

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_Init.
Value	0x00U

4.2.2.29. CRYPTO[_<VI>_<AI>]_SID_KEYCOPY

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyCopy.
Value	0x10U

4.2.2.30. CRYPTO[_<VI>_<AI>]_SID_KEYDERIVE

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyDerive.
Value	0x08U

4.2.2.31. CRYPTO[_<VI>_<AI>]_SID_KEYELEMENTCOPY

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyElementCopy.
Value	0x0FU

4.2.2.32. CRYPTO[_<VI>_<AI>]_SID_KEYELEMENTGET

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyElementGet.
Value	0x06U

4.2.2.33. CRYPTO[_<VI>_<AI>]_SID_KEYELEMENTIDSGET

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyElementIdsGet.

Value	0x11U

4.2.2.34. CRYPTO[_<VI>_<AI>]_SID_KEYELEMENTSET

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyElementSet.
Value	0x04U

4.2.2.35. CRYPTO[_<VI>_<AI>]_SID_KEYEXCHANGECALCPUBVAL

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyExchangeCalcPubVal.
Value	0x09U

4.2.2.36. CRYPTO[_<VI>_<AI>]_SID_KEYEXCHANGECALCSECRET

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyExchangeCalcSecret.
Value	0x0AU

4.2.2.37. CRYPTO[_<VI>_<AI>]_SID_KEYGENERATE

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyGenerate.
Value	0x07U

4.2.2.38. CRYPTO[_<VI>_<AI>]_SID_KEYVALIDSET

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_KeyValidSet.
Value	0x05U

4.2.2.39. CRYPTO[_<VI>_<AI>]_SID_MAINFUNCTION

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_MainFunction.
Value	0x0CU

4.2.2.40. CRYPTO[_<VI>_<AI>]_SID_PROCESSJOB

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_ProcessJob.
Value	0x03U

4.2.2.41. CRYPTO[_<VI>_<AI>]_SID_RANDOMSEED

Purpose	AUTOSAR API service ID for Crypto[_<Vi>_<Ai>]_RandomSeed.
Value	0x0DU

4.2.2.42. CRYPTO[_<VI>_<AI>]_SIGNATURE_ALGORITHM_ECC

Purpose	Macros for the signature algorithm type EDDSA used in a certificate.
Value	1U

4.2.2.43. CRYPTO[_<VI>_<AI>]_SIGNATURE_ALGORITHM_RSA

Purpose	Macros for the signature algorithm type RSA used in a certificate.
Value	2U

4.2.2.44. CRYPTO[_<VI>_<AI>]_WA_ALLOWED

Purpose	Key element can be written in plaintext.
Value	0x03U

4.2.2.45. CRYPTO[_<VI>_<AI>]_WA_DENIED

Purpose	The key element can not be written from outside the crypto driver.
Value	0x01U

4.2.2.46. CRYPTO[_<VI>_<AI>]_WA_ENCRYPTED

Purpose	Key element can be written encrypted.

Value	0x04U

4.2.2.47. CRYPTO[_<VI>_<AI>]_WA_INTERNAL_COPY

Purpose	Key element within the same crypto driver can be copied to the key element.
Value	0x02U

4.2.3. Objects

4.2.3.1. Crypto[_<Vi>_<Ai>]_DriverObjects

Purpose	Array of driver objects.
Type	Crypto[_<Vi>_<Ai>]_DriverObject

4.2.3.2. Crypto[_<Vi>_<Ai>]_Initialized

Purpose	Module initialization status.
Type	`boolean`

4.2.3.3. Crypto[_<Vi>_<Ai>]_KeyElements

Purpose	Array of key elements.
Type	Crypto[_<Vi>_<Ai>]_KeyElement

4.2.3.4. Crypto[_<Vi>_<Ai>]_Keys

Purpose	Array of keys.
Type	Crypto[_<Vi>_<Ai>]_Key

4.2.3.5. Crypto[_<Vi>_<Ai>]_Queues

Purpose	Array of queues.

Type	Crypto[_<Vi>_<Ai>]_QueueType

4.2.4. Functions

4.2.4.1. Crypto[_<Vi>_<Ai>]_CancelJob

Purpose	Cancel function of the crypto driver.	
Synopsis	Std_ReturnType **Crypto[_<Vi>_<Ai>]_CancelJob** (uint32 objectId , Crypto_JobType * job);	
Sync/Async	Synchronous	
Reentrancy	Nonreentrant	
Parameters (in)	objectId	Identifier of the crypto driver object that processes the job.
	job	Reference to the job that shall be cancelled.
Return Value	Result of the job cancellation attempt.	
	E_OK	Job has been removed from the queue.
	E_NOT_OK	Job could not be cancelled.
	CRYPTO_E_CANCELED	Active job has been successfully canceled.
Description	This function is used to cancel a requested job. If the job is currently in the queue of the passed driver object, this function will attempt to remove the job from the queue. If the job is currently processed by the crypto engine, the function will pass the cancellation request down to the process layer.	

4.2.4.2. Crypto[_<Vi>_<Ai>]_CertificateParse

Purpose	Parse a certificate.	
Synopsis	Std_ReturnType **Crypto[_<Vi>_<Ai>]_CertificateParse** (uint32 CryptoKeyId);	
Parameters (in)	CryptoKeyId	Identifier of the key that contains the certificate
Return Value	Result of the request	

	E_OK	Request successful
	E_NOT_OK	Request failed
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.3. Crypto[_<Vi>_<Ai>]_CertificateVerify

Purpose	Verify a certificate.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_CertificateVerify`** `( uint32 CryptoKeyId , uint32 VerifyCryptoKeyId , Crypto_VerifyResultType * VerifyPtr );`	
Parameters (in)	`CryptoKeyId`	Identifier of the key that shall be used for verification
	`VerifyCryptoKeyId`	Identifier of the key that contains the certificate
Parameters (out)	`VerifyPtr`	Pointer to the memory location where the result of the verification shall be stored
Return Value	Result of the request	
	E_OK	Request successful
	E_NOT_OK	Request failed
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.4. Crypto[_<Vi>_<Ai>]_GetVersionInfo

Purpose	Retrieve version info for the crypto driver module.	
Synopsis	`void` **`Crypto[_<Vi>_<Ai>]_GetVersionInfo`** `( Std_VersionInfoType * versioninfo );`	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (out)	`versioninfo`	Pointer to the VersionInfo structure that shall be filled with data.
Description	This function copies the version information to the passed VersionInfo struct.	

4.2.4.5. Crypto[_<Vi>_<Ai>]_Init

Purpose	Init function of the Crypto Driver.
Synopsis	`void` **`Crypto[_<Vi>_<Ai>]_Init`** `( void );`
Sync/Async	Synchronous
Reentrancy	Nonreentrant
Description	This function initializes the AUTOSAR layer and forwards the initialization request to the process layer.

4.2.4.6. Crypto[_<Vi>_<Ai>]_KeyCopy

Purpose	Copy all key elements of a key.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyCopy`** `( uint32 CryptoKeyId , uint32 TargetCryptoKeyId );`	
Parameters (in)	`CryptoKeyId`	Identifier of the key whose elements shall be copied
	`TargetCryptoKeyId`	Identifier of the destination key
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
	`CRYPTO_E_KEY_READ_FAIL`	Read access was denied
	`CRYPTO_E_KEY_WRITE_FAIL`	Write access was denied
	`CRYPTO_E_KEY_NOT_AVAILABLE`	The requested key is not available
	`CRYPTO_E_KEY_SIZE_MISMATCH`	Key element sizes are not compatible
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.7. Crypto[_<Vi>_<Ai>]_KeyDerive

Purpose	Derive a key from another key.

Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyDerive`** `( uint32 CryptoKeyId , uint32 TargetCryptoKeyId );`	
Parameters (in)	`CryptoKeyId`	Identifier of the crypto key that shall be used for derivation
	`TargetCryptoKeyId`	Identifier of the crypto key where the derived key shall be stored
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.8. Crypto[_<Vi>_<Ai>]_KeyElementCopy

Purpose	Copy a key element.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyElementCopy`** `( uint32 CryptoKeyId , uint32 KeyElementId , uint32 TargetCryptoKeyId , uint32 TargetKeyElementId );`	
Parameters (in)	`CryptoKeyId`	Identifier of the key whose element shall be copied
	`KeyElementId`	Identifier of the key element that shall be copied
	`TargetCryptoKeyId`	Identifier of the destination key
	`TargetKeyElementId`	Identifier of the destination key element
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
	`CRYPTO_E_BUSY`	Request failed, crypto driver object is busy
	`CRYPTO_E_KEY_READ_FAIL`	Read access was denied
	`CRYPTO_E_KEY_WRITE_FAIL`	Write access was denied
	`CRYPTO_E_KEY_NOT_AVAILABLE`	The requested key is not available
	`CRYPTO_E_KEY_SIZE_MISMATCH`	Key element sizes are not compatible
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.9. Crypto[_<Vi>_<Ai>]_KeyElementGet

Purpose	Retrieve the value of a key element.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyElementGet`** `( uint32 CryptoKeyId , uint32 KeyElementId , uint8 * ResultPtr , uint32 * ResultLengthPtr );`	
Parameters (in)	`CryptoKeyId`	Identifier of the crypto key whose element shall be retrieved
	`KeyElementId`	Identifier of the key element that shall be retrieved
Parameters (in,out)	`ResultLengthPtr`	Pointer to the location where the length information is stored. Shall contain the amount of bytes that shall be read. After finishing the request, it contains the amount of bytes that has been stored in the result.
Parameters (out)	`ResultPtr`	Pointer to the memory location where the key element data shall be stored
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
	`CRYPTO_E_KEY_READ_FAIL`	Read access was denied
	`CRYPTO_E_KEY_NOT_AVAILABLE`	The requested key is not available
	`CRYPTO_E_SMALL_BUFFER`	The provided buffer is too small to store the result
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.10. Crypto[_<Vi>_<Ai>]_KeyElementIdsGet

Purpose	Get the Ids of the key elements available within the requested key.
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyElementIdsGet`** `( uint32 CryptoKeyId , uint32 * KeyElementIdsPtr , uint32 * KeyElementIdsLengthPtr );`

Parameters (in)	`CryptoKeyId`	Identifier of the crypto key whose key element ids shall be retrieved
Parameters (in,out)	`KeyElementIdsLengthPtr`	Memory location, where the number of key elements shall be stored. On calling this function it shall contain the number of element Ids that can be stored within the provided buffer.
Parameters (out)	`KeyElementIdsPtr`	Memory location, where the key element ids shall be stored
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
	`CRYPTO_E_SMALL_BUFFER`	The provided buffer is too small to store the result
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.11. Crypto[_<Vi>_<Ai>]_KeyElementSet

Purpose	Set a key element.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyElementSet`** `( uint32 CryptoKeyId , uint32 KeyElementId , const uint8 * KeyPtr , uint32 KeyLength );`	
Parameters (in)	`CryptoKeyId`	Identifier of the crypto key whose key element shall be set
	`KeyElementId`	Identifier of the key element that shall be set
	`KeyPtr`	Pointer to the key data that shall be set as the key element
	`KeyLength`	Length of the key element in bytes
Return Value	Result of the key setting operation	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
	`CRYPTO_E_BUSY`	Request failed, crypto driver object is busy
	`CRYPTO_E_KEY_WRITE_FAIL`	Write access was denied

	CRYPTO_E_KEY_NOT_AVAILABLE	The requested key is not available
	CRYPTO_E_KEY_SIZE_MISMATCH	Provided data size does not match key element size
Description	This function checks the provided parameters and forwards the key setting request to the process layer	

4.2.4.12. Crypto[_<Vi>_<Ai>]_KeyExchangeCalcPubVal

Purpose	Calculate the public value for the key exchange.	
Synopsis	`Std_ReturnType Crypto[_<Vi>_<Ai>]_KeyExchangeCalcPubVal ( uint32 CryptoKeyId , uint8 * PublicValuePtr , uint32 * PublicValueLengthPtr );`	
Parameters (in)	CryptoKeyId	Identifier of the crypto key that shall be used for the key exchange
Parameters (in,out)	PublicValueLengthPtr	Pointer to the memory location where the length information shall be stored. On calling this function, this location shall contain the size of the provided buffer.
Parameters (out)	PublicValuePtr	Pointer to the memory location where the public value shall be stored
Return Value	Result of the request	
	E_OK	Request successful
	E_NOT_OK	Request failed
	CRYPTO_E_SMALL_BUFFER	The provided buffer is too small to store the result
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.13. Crypto[_<Vi>_<Ai>]_KeyExchangeCalcSecret

Purpose	Calculate the shared secret for the key exchange.
Synopsis	`Std_ReturnType Crypto[_<Vi>_<Ai>]_KeyExchangeCalcSecret ( uint32 CryptoKeyId , const uint8 * PartnerPublicValuePtr , uint32 PartnerPublicValueLength );`

Parameters (in)	`CryptoKeyId`	Identifier of the crypto key that shall be used for the key exchange
	`PartnerPublicValueLength`	Length of the partner's public value in bytes
Parameters (out)	`PartnerPublicValuePtr`	Pointer to the memory where the public value of the partner is located
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
	`CRYPTO_E_SMALL_BUFFER`	The provided buffer is too small to store the result
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.14. Crypto[_<Vi>_<Ai>]_KeyGenerate

Purpose	Generate a new key.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyGenerate`** `( uint32 CryptoKeyId );`	
Parameters (in)	`CryptoKeyId`	Identifier of the crypto key for which key material shall be generated
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.15. Crypto[_<Vi>_<Ai>]_KeyValidSet

Purpose	Set a key to valid.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_KeyValidSet`** `( uint32 CryptoKeyId );`	
Parameters (in)	`CryptoKeyId`	Identifier of the crypto key that shall be set to valid

Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
Description	This function checks the provided parameters and forwards the request to the process layer	

4.2.4.16. Crypto[_<Vi>_<Ai>]_MainFunction

Purpose	Cyclic main function of the crypto driver.
Synopsis	`void Crypto[_<Vi>_<Ai>]_MainFunction ( void );`
Sync/Async	Synchronous
Reentrancy	Nonreentrant
Description	This function checks all available queues for jobs to be processed. If a driver object is idle and has jobs in its queue, they are passed to the process layer.

4.2.4.17. Crypto[_<Vi>_<Ai>]_ProcessJob

Purpose	Process function of the crypto driver.	
Synopsis	`Std_ReturnType Crypto[_<Vi>_<Ai>]_ProcessJob ( uint32 objectId , Crypto_JobType * job );`	
Sync/Async	Synchronous	
Reentrancy	Nonreentrant	
Parameters (in)	`objectId`	Identifier of the crypto driver object that shall process the job.
	`job`	Reference to the job that shall be processed.
Return Value	Result of the job processing.	
	`E_OK`	Job has been put into the queue or successfully processed.
	`E_NOT_OK`	Job could not be processed.
	`CRYPTO_E_BUSY`	Request failed, the crypto driver object is busy.

	CRYPTO_E_KEY_NOT_VALID	Request failed, the key to be used 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 of the crypto driver object is full.
	CRYPTO_E_ENTROPY_EXHAUSTION	Request failed, the entropy is exhausted.
	CRYPTO_E_SMALL_BUFFER	Request failed, the provided buffer is too small to store the result.
	CRYPTO_E_JOB_CANCELED	Request failed, the synchronous job was canceled.
Description	This function accepts the process requests. It checks the parameters for correctness and dispatches according to service, algorithm family and algorithm mode of the requested job. If the requested driver object supports the requested primitive, the function passes the request to the process layer, if the crypto driver object is not busy. If the driver object is busy, the function indicates an error (synchronous processing) or attempts to put the job into the queue. (asynchronous processing)	

4.2.4.18. Crypto[_<Vi>_<Ai>]_ProcessJob_Dispatch

Purpose	Process function of the crypto driver.	
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_ProcessJob_Dispatch`** `( uint32 objectId , Std_ReturnType RetVal , Crypto_JobType * job );`	
Return Value		
Description	This function accepts the process requests. It checks the parameters for correctness and dispatches according to service, algorithm family and algorithm mode of the requested job. If the requested driver object supports the requested primitive, the function passes the request to the process layer, if the crypto driver object is not busy. If the driver object is busy, the function indicates an error (synchronous processing) or attempts to put the job into the queue. (asynchronous processing)	

4.2.4.19. Crypto[_<Vi>_<Ai>]_RandomSeed

Purpose	Generate internal seed state for requested key.
Synopsis	`Std_ReturnType` **`Crypto[_<Vi>_<Ai>]_RandomSeed`** `( uint32 CryptoKeyId , const uint8 * EntropyPtr , uint32 EntropyLength );`

Parameters (in)	`CryptoKeyId`	Identifier of the crypto key for which a seed shall be generated
	`EntropyPtr`	Pointer to the memory location which contains the entropy data
	`EntropyLength`	Length of the entropy in bytes
Return Value	Result of the request	
	`E_OK`	Request successful
	`E_NOT_OK`	Request failed
Description	This function checks the provided parameters and forwards the request to the process layer	

4.3. Integration notes

4.3.1. Integration requirements

WARNING

Integration requirements list is not exhaustive

The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user's guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product.

4.3.1.1. Crypto.Req.Integration_CryptoInit

Description	Crypto_Init() shall be called during the start-up procedure of the ECU before any other API of the module is called.
Rationale	

4.3.1.2. Crypto.Req.Integration_StartupNvMRead

Description	If the use of a NvM block is enabled by setting the parameter CryptoKeyElementPersist, the Crypto_Init() shall be called after the NvM module is initialized and the NvM_ReadAll job is successfully completed. If your application does not enable NvMSelect-

	BlockForReadAll for persistent key elements, ensure that the corresponding blocks are read before the Crypto module is initialized.
Rationale	The Crypto module does not call NvM_ReadBlock() to load key element values during startup.

4.3.1.3. Crypto.Req.Integration_CertificateVerify_Primitive

Description	The primitive that shall be used to verify the certificate shall be configured and referenced in the driver object.
Rationale	Certificate verify uses a primitive to verify the signature of the certificate. For this the integrator shall reference that particular primitive in the driver object.

4.3.1.4. Crypto.Req.Integration_CertificateVerify_CurrentDate

Description	If the optional key element CryptoKeyElement_CRYPTO_KE_CERTIFICATE_CURRENT_TIME (KeyElementId 19U) is contained in the certicate key type, the current time verification is turned on. It is up to the integrator to set a valid current date before verifing the certificate. The date format for the key element is YYMMDD. For example 4th of Feburary 2001 would be formated as 000100020004 (6 bytes).
Rationale	The key element CryptoKeyElement_CRYPTO_KE_CERTIFICATE_CURRENT_TIME (KeyElementId 19U) contains the current date. During certificate verification, this date is used to verify whether the current time is within the effective and expiry date of the certificate that is being verified. Therefore the current date must be set by the integrator before a certificate is verified.

4.3.1.5. Crypto.Req.Integration_CertificateParse_NeededKeyElements

Description	If the certificate management is used, the key holding the certificate data shall at least have the following key elements: CRYPTO_KE_CERTIFICATE_DATA (KeyElementId 0U), CRYPTO_KE_CERTIFICATE_SIGNATURE (KeyElementId 28U), CRYPTO_KE_CERTIFICATE_SUBJECT (KeyElementId 26U), CRYPTO_KE_-CERTIFICATE_SUBJECT_PUBLIC_KEY (KeyElementId 1U), CRYPTO_KE_-CERTIFICATE_SIGNATURE_ALGORITHM (KeyElementId 22U), CRYPTO_KE_-CERTIFICATE_PARSING_FORMAT (KeyElementId 18U), CRYPTO_KE_CERTIFICATE_VALIDITY_NOT_AFTER (KeyElementId 25U), CRYPTO_KE_CERTIFICATE_VALIDITY_NOT_BEFORE (KeyElementId 24U), CRYPTO_KE_CERTIFICATE_CURRENT_TIME (KeyElementId 19U) (optional).

Rationale	To successfully parse a certificate, the above mentioned key elements are needed. Key elements CRYPTO_KE_CERTIFICATE_DATA, and CRYPTO_KE_CERTIFICATE_PARSING_FORMAT are needed as input for the certificate parse operation, while the others are used to store the information retrieved from the certificated. Please note that CRYPTO_KE_CERTIFICATE_CURRENT_TIME (KeyElementId 19U) is an optional key element which needs to be handled by the integrator.

4.3.1.6. Crypto.Req.Integration_CertificateParse_KeyElements

Description	If the certificate management is used, the key elements CRYPTO_KE_CERTIFICATE_DATA, CRYPTO_KE_CERTIFICATE_SIGNATURE, CRYPTO_KE_CERTIFICATE_SUBJECT and CRYPTO_KE_CERTIFICATE_SUBJECT_PUBLIC_KEY should be configured large enough for all the certificates.
Rationale	Since different certificates will use the same key types and key elements, the individual key elements must be large enough for all the certificates.

4.3.1.7. Crypto.Req.Integration_CertificateParse_ParseFormatCVC

Description	If the provided CV certificate is base64 encoded, the key element CRYPTO_KE_CERTIFICATE_PARSING_FORMAT shall be set to 0x08U (CRYPTO_KE_FORMAT_BIN_CERT_CVC).
Rationale	The CV certificate may be passed as plain text or as base64 encoded. If the provided certificate is base64 encoded, then the key element CRYPTO_KE_CERTIFICATE_PARSING_FORMAT shall be set to 0x08U (CRYPTO_KE_FORMAT_BIN_CERT_CVC). This allows the API to decode the certificate from base64 before processing it. For a CV certificate passed as plain text, CRYPTO_KE_CERTIFICATE_PARSING_FORMAT shall be set to 0x00 (default initial value of the CRYPTO_KE_CERTIFICATE_PARSING_FORMAT key element).

4.3.1.8. Crypto.Req.Integration_CertificateParse_ParseFormatCVCPlainText

Description	If the key element CRYPTO_KE_CERTIFICATE_PARSING_FORMAT is set to 0x00U (default initial value of the CRYPTO_KE_CERTIFICATE_PARSING_FORMAT key element), the provided certificate shall be considered to be in plain text, and no base64 decoding would be performed.
Rationale	In order to make sure that the certificates are interpreted correctly, key element CRYPTO_KE_CERTIFICATE_PARSING_FORMAT is used. Since, the certificates are

	only CV certificates, key element CRYPTO_KE_CERTIFICATE_PARSING_FORMAT is used to denote their format i.e base64 encoded or plain text.

4.3.1.9. Crypto.Req.Integration_CertificateParse_SignatureAlgorithm

Description	If using certificate management, key element CRYPTO_KE_CERTIFICATE_SIGNATURE_ALGORITHM must be configured. Certificate parse function returns E_NOT_OK is this key element is not configured.
Rationale	This enables quick access of the signature algorithm during certificate verification.

4.3.1.10. Crypto.Req.Integration_IntGCM

Description	The additional authenticated data(AAD) must be provided with and only with the first call of Update.
Rationale	The additional authenticated data must be processed before the plaintext/cyphertext is received.

4.3.1.11. Crypto.Req.Integration_BuildForMultiInstances

Description	The build process must not consider source files from the Crypto plugin folder, but instead shall consider the output directory, particularly the output\generated\instance\include and the output\generated\instance\src directories.
Rationale	The source files in the Crypto plugin folder are prepared for multiple instantiation. They will be copied, processed and renamed during project's generation. For further information please see the Crypto module user's guide.

4.3.1.12. Crypto.Req.Integration_CompilerCfgForMultiInstances

Description	In case multiple instances of the Crypto module are used, all required memory and pointer classes as specified by AUTOSAR (e.g. CRYPTO_<vi>_<ai>_CODE, CRYPTO_<vi>_<ai>_APPL_DATA, etc.) have to be provided by the integrator. To simplify this process, appropriate template files Compiler_CfgExt_Crypto_<vi>_<ai>.template.h per Crypto module instance are provided in output\generated\templates. It is up to the integrator either to merge/include all these files into Compiler_Cfg.h or to in-

	clude all of them in one Compiler_CfgExt.h file that can be included in Compiler_Cfg.h by defining the macro COMPILERCFG_EXTENSION_FILE in the build process.
Rationale	The proposed process is specified in AUTOSAR_SWS_CompilerAbstraction specification. For further information on the usage of the COMPILERCFG_EXTENSION_FILE macro see the Tresos documentation.

4.3.1.13. Crypto.Req.Integration_CMACkeyPreCalc

Description	In order to make use of the optimization regarding the precalculation of expanded key, K1, and K2, there shall be three new key elements configured for a MAC keytype. The sizes of these key elements shall be as follows: expanded key - 240U bytes, K1 - 16U bytes, K2 - 16U bytes.
Rationale	These elements will be calculated when the AES-CMAC key element of this MAC keytype is set.

4.3.1.14. Crypto.Req.Integration_RandomSeedAlgorithm

Description	The keys used with the key management function Crypto_RandomSeed need to include a CRYPTO_KE_RANDOM_ALGORITHM key element with the keyElementId 4 and keyElementSize 1. To allow the Crypto_RandomSeed function to choose the correct seed function, this key element has to be set to the random primitives algorithm id according to the following list: 0x00 CryptoPrimitive_SSG_Random 0x01 CryptoPrimitive_CTRDRBG_Random
Rationale	The key management function RandomSeed is independent from the jobs and needs to be configured.

4.3.1.15. Crypto.Req.Integration_KeyDeriveFunction

Description	The keys used with the key management function Crypto_xVIx_xAIx_KeyDerive need to include a CRYPTO_KE_KEYDERIVATION_ALGORITHM key element with the keyElementId 15 and keyElementSize 1. To allow the Crypto_xVIx_xAIx_KeyDerive function to choose the correct pseudorandom function, this key element has to be set to the pseudorandom primitives algorithm id according to the following list: 0x02 CryptoPrimitive_SHA256_HMAC_Mac_Generate 0x04 CryptoPrimitive_SHA2_256_Hash
Rationale	The key management function Crypto_xVIx_xAIx_KeyDerive is independent from the jobs and needs to be configured.

4.3.1.16. Crypto.Req.Integration_KeyElementCopy_invalidKey

Description	The function Crypto_xVIx_xAIx_KeyElementCopy copies the source key element from the source key to the target key element of the target key even if the source key is currently invalid. It is the task of the integrator to make sure that the key that the target key is set to valid after the key element is copied. On the other hand the Crypto_xVIx_xAIx_KeyCopy function would not copy the key from source to destination if the source key is invalid. No processing would take place and E_NOT_OK would be returned.
Rationale	The usage of the Crypto_xVIx_xAIx_KeyElementCopy should not be limited by limiting the source keys to be valid keys only.

4.3.1.17. Crypto.Req.Integration_SymKeyType

Description	The Crypto module defines Crypto_xVIx_xAIx_SymKeyType as Csm_SymKeyType if Csm_SymKeyType exists in Csm. It is however up to the integrator to make sure that if it exists in the Csm, CsmSymKeyMaxLength is configured large enough to be used for all the keys in Crypto.
Rationale	If the user configures CsmSymKeyMaxLength in Csm, we should use the created data type Csm_SymKeyType, and the user should make sure that it is properly configured.

4.3.1.18. Crypto.Req.Integration_SymKeyType_KeyMaxLength

Description	If CsmSymKeyMaxLength is disabled, the Crypto module shall implement its internal typedef for Crypto_xVIx_xAIx_SymKeyType, using the length of the largest configured key element with Id 1.
Rationale	If the Csm does not declare the Csm_SymKeyType, Crypto should declare independent internal Crypto_xVIx_xAIx_SymKeyType. This shall avoid unnecessary dependence on Csm.

4.3.1.19. Crypto.Req.Integration_SymKeyType_NoKey

Description	If no key is configured in the Crypto or no key is configured with an element that has Id 1, Crypto_xVIx_xAIx_SymKeyType shall not be created. If CTRDRBG is configured (for random number generation), Crypto_xVIx_xAIx_SymKeyType would be created with length 32 bytes.

Rationale	Since CTRDRBG needs a key for internal usage, Crypto_xVIx_xAIx_SymKeyType shall be created with minimum length 32 bytes.

4.3.1.20. Crypto.Req.Integration_AsymPrivateKeyType

Description	The Crypto module defines Crypto_xVIx_xAIx_AsymPrivateKeyType as Csm_AsymPrivateKeyType if Csm_AsymPrivateKeyType exists in Csm. It is however up to the integrator to make sure that if it exists in the Csm, CsmAsymPrivateKeyMaxLength is configured large enough to be used for all the keys in Crypto.
Rationale	If the user configures CsmAsymPrivateKeyMaxLength in Csm, we should use the created data type Csm_AsymPrivateKeyType, and the user should make sure that it is properly configured.

4.3.1.21. Crypto.Req.Integration_AsymPrivateKeyType_KeyMaxLength

Description	If Csm_AsymPrivateKeyType does not exist in Csm, a type Crypto_xVIx_xAIx_AsymPrivateKeyType shall be created in Crypto module, using the length of the largest key element with Id 1, referenced in the keys.
Rationale	Define internal type if the type does not exist in Csm module.

4.3.1.22. Crypto.Req.Integration_AsymPrivateKeyType_NoKey

Description	If no key is configured in the Crypto or no key is configured with an element that has Id 1, Crypto_xVIx_xAIx_AsymPrivateKeyType shall not be created.
Rationale	Crypto_xVIx_xAIx_AsymPrivateKeyType is not needed if no key is configured and no primitive uses it.

4.3.1.23. Crypto.Req.Integration_AsymPublicKeyType

Description	The Crypto module defines Crypto_xVIx_xAIx_AsymPublicKeyType as Csm_AsymPublicKeyType if Csm_AsymPublicKeyType exists in Csm. It is however up to the integrator to make sure that if it exists in the Csm, CsmAsymPublicKeyMaxLength is configured large enough to be used for all the keys in Crypto.

Rationale	If the user configures CsmAsymPublicKeyMaxLength in Csm, we should use the created data type Csm_AsymPublicKeyType, and the user should make sure that it is properly configured.

4.3.1.24. Crypto.Req.Integration_AsymPublicKeyType_KeyMaxLength

Description	If Csm_AsymPublicKeyType does not exist in Csm, a type Crypto_xVIx_xAIx_AsymPublicKeyType shall be created in Crypto module, using the length of the largest key element with Id 1, referenced in the keys.
Rationale	Define internal type if the type does not exist in Csm module.

4.3.1.25. Crypto.Req.Integration_AsymPublicKeyType_NoKey

Description	If no key is configured in the Crypto or no key is configured with an element that has Id 1, Crypto_xVIx_xAIx_AsymPublicKeyType shall not be created.
Rationale	Crypto_xVIx_xAIx_AsymPublicKeyType not needed if no key is configured and no primitive uses it.

4.3.1.26. Crypto.Req.Integration_Signature_EdDSA

Description	The following configuration is required to use the Ed25519ph primitive for the SIGNATUREGENERATE/VERIFY service: CryptoPrimitiveAlgorithmFamiliy CRYPTO_ALGOFAM_ED25519, CryptoPrimitiveAlgorithmMode CRYPTO_ALGOMODE_NOT_SET, CryptoPrimitiveAlgorithmSecondaryFamiliy CRYPTO_ALGOFAM_SHA2_512.
Rationale	The provided EdDSA primitive for SIGNATUREGENERATE/VERIFY is the implementation of the Ed25519ph. It is specified in the Crypto_preconf.xdm file.

4.3.1.27. Crypto.Req.Integration_RsaesOaepEncryptionKeyFormat

Description	To use the RSA with Optimal Asymmetric Encryption Padding (RSAES-OAEP) encryption of the Crypto Driver correctly, the modulus and the exponent of the public RSA key have to be DER-Encoded before they are stored inside the key element CRYPTO_KE_CIPHER_KEY. The ASN.1 type of the encoded key shall be as follows: RSAPublicKey ::= SEQUENCE { modulus INTEGER -- n, publicExponent INTEGER -- e }

Rationale	Multiple elements are stored in one single key element. The DER-Encoding is used to be able to separate the elements in the Crypto Driver.

4.3.1.28. Crypto.Req.Integration_RsaesOaepEncryptionAdditionalInput

Description	If an additional input A shall be used in the RSA with Optimal Asymmetric Encryption Padding (RSAES-OAEP) encryption, then it has to be stored as byte array in the key element CRYPTO_KE_ADDITIONAL_INPUT with id 1004U.
Rationale	The additional input A is an optional parameter to the RSAES-OAEP algorithm. If the key element does not exist, the additional input A will be an empty string.

4.3.1.29. Crypto.Req.Integration_RsaesOaepEncryptionRandomSeed

Description	To use the RSA with Optimal Asymmetric Encryption Padding (RSAES-OAEP) encryption, a key element CRYPTO_KE_RANDOM_SEED_STATE is required in the key. This key element should be seeded before the first use by using the key management function Crypto_RandomSeed().
Rationale	The AES-CTR_DRBG random number generator is used during the RSAES-OAEP encryption and needs a valid internal seed state to generate random numbers.

4.3.1.30. Crypto.Req.Integration_RsaesOaepDecryptionKeyFormat

Description	To use the RSA with Optimal Asymmetric Encryption Padding (RSAES-OAEP) decryption of the Crypto Driver correctly, the modulus and the exponent of the private RSA key have to be DER-Encoded before they are stored inside the key element CRYPTO_KE_CIPHER_KEY. The ASN.1 type of the encoded key shall be as follows: RSAPrivateKey ::= SEQUENCE { modulus INTEGER -- n, privateExponent INTEGER -- d }
Rationale	Multiple elements are stored in one single key element. The DER-Encoding is used to be able to separate the elements in the Crypto Driver.

4.3.1.31. Crypto.Req.Integration_RsaesOaepDecryptionAdditionalInput

Description	If an additional input A shall be used in the RSA with Optimal Asymmetric Encryption Padding (RSAES-OAEP) decryption, then it has to be stored as byte array in the key element CRYPTO_KE_ADDITIONAL_INPUT with id 1004U.

Rationale	The additional input A is an optional parameter to the RSAES-OAEP algorithm. If the key element does not exist, the additional input A will be an empty string.

4.3.1.32. Crypto.Req.Integration_RsaSsaPkcs1V1_5SignatureGenerationKeyFormat

Description	To use the RSASSA-PKCS1-v1_5 signature generation of the Crypto Driver correctly, the modulus and the exponent of the private RSA key have to be DER-Encoded before they are stored inside the key element CRYPTO_KE_CIPHER_KEY. The ASN.1 type of the encoded key shall be as follows: RSAPrivateKey ::= SEQUENCE { modulus INTEGER -- n, privateExponent INTEGER -- d }
Rationale	Multiple elements are stored in one single key element. The DER-Encoding is used to be able to separate the elements in the Crypto Driver.

4.3.1.33. Crypto.Req.Integration_RsaSsaPkcs1V1_5SignatureVerificationKeyFormat

Description	To use the RSASSA-PKCS1-v1_5 signature verification of the Crypto Driver correctly, the modulus and the exponent of the public RSA key have to be DER-Encoded before they are stored inside the key element CRYPTO_KE_CIPHER_KEY. The ASN.1 type of the encoded key shall be as follows: RSAPublicKey ::= SEQUENCE { modulus INTEGER -- n, publicExponent INTEGER -- e }
Rationale	Multiple elements are stored in one single key element. The DER-Encoding is used to be able to separate the elements in the Crypto Driver.

4.3.1.34. Crypto.Req.Integration_RsaSsaPssSignatureVerificationKeyFormat

Description	To use the RSASSA-PSS signature verification of the Crypto Driver correctly, the modulus and the exponent of the public RSA key have to be DER-Encoded before they are stored inside the key element CRYPTO_KE_CIPHER_KEY. The ASN.1 type of the encoded key shall be as follows: RSAPublicKey ::= SEQUENCE { modulus INTEGER -- n, publicExponent INTEGER -- e }
Rationale	Multiple elements are stored in one single key element. The DER-Encoding is used to be able to separate the elements in the Crypto Driver.

4.3.1.35. Crypto.Req.Integration_ECDH_Algorithm

Description	To use the ECDH key exchange of the Crypto Driver correctly, i.e. to choose between the x25519 and the NIST ECC elliptic curves, the algorithm family has to be pro-

	vided. Thus, the given key has to contain the key element CRYPTO_KE_KEYEXCHANGE_ALGORITHM and it must be set to CRYPTO_ALGOFAM_ED25519 or to CRYPTO_ALGOFAM_ECCNIST accordingly.
Rationale	After extending the ECDH key exchange for another elliptic curve, the differentiation of the concrete algorithm is required.

4.3.1.36. Crypto.Req.Integration_ECDH_ECCNISTCurveConfig

Description	To use the ECDH key exchange of the Crypto Driver correctly for use with NIST ECC elliptic curves, the concrete elliptic curve has to be provided. Thus, the given key has to contain the key element CRYPTO_KE_KEYEXCHANGE_CURVE (CryptoKeyElementId 30) and it must be set for secp256r1 (NIST P-256) or secp384r1 (NIST P-384). The configuration is done by using the OID of the curves, secp256r1: 1.2.840.10045.3.1.1.7 and secp384r1: 1.3.132.0.34. The Crypto Driver uses the DER encoded value of the OID. Therefore, CRYPTO_KE_KEYEXCHANGE_CURVE has to be set to '06 08 2a 86 48 ce 3d 03 01 07' for secp256r1, and to '06 05 2b 81 04 00 22' for secp384r1.
Rationale	The AUTOSAR SWS does not specify how to configure the curve to be chosen in key exchange with ECC NIST curves. A proposed solution using the above configuration has been entered to https://jira.autosar.org/browse/AR-3030.

5. Bibliography

Bibliography

[1] *AUTOSAR Specification of Crypto Service Manager*, Issue 4.3.0, Publisher: AUTOSAR

[2] *AUTOSAR Specification of Crypto Interface*, Issue 4.3.0, Publisher: AUTOSAR

[3] *AUTOSAR Specification of BSW Module Description Template*, Issue 4.3.0, Publisher: AUTOSAR

[4] *AUTOSAR Specification of Crypto Driver*, Issue 4.3.0, Publisher: AUTOSAR