| Document Title | Requirements on Crypto Stack |
|---|---|
| **Document Owner** | AUTOSAR |
| **Document Responsibility** | AUTOSAR |
| **Document Identification No** | 426 |

| | |
|---|---|
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R24-11 |

**Document Change History**

| Date | Release | Changed by | Description |
|---|---|---|---|
| 2024-11-27 | R24-11 | AUTOSAR Release Management | • No content changes |
| 2023-11-23 | R23-11 | AUTOSAR Release Management | • No content changes |
| 2022-11-24 | R22-11 | AUTOSAR Release Management | • Add support fo certificate formats CVC and X.509 |
| 2021-11-25 | R21-11 | AUTOSAR Release Management | • Editorial changes |
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • Updated supported algorithms |
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • Updated items in glossary and abbreviation list<br>• Changed Document Status from Final to published |
| 2018-10-31 | 4.4.0 | AUTOSAR Release Management | • Adding Coverage of Key Manager<br>• Removed Secure Counter functionality<br>• Editorial changes |
| 2017-12-08 | 4.3.1 | AUTOSAR Release Management | • Default error detection renamed to development error detection<br>• Editorial changes |

△

| | | | |
|---|---|---|---|
| 2016-11-30 | 4.3.0 | AUTOSAR Release Management | • Added requirements for the whole Crypto Stack and renamed the document<br>• Introduced crypto job concept<br>• Introduced key management concept |
| 2014-10-31 | 4.2.1 | AUTOSAR Release Management | • Editorial changes |
| 2013-10-31 | 4.1.2 | AUTOSAR Release Management | • Editorial changes |
| 2013-03-15 | 4.1.1 | AUTOSAR Administration | • [TPS_STDT_0078] formatting<br>• Traceability of BSWAndRTE_Features |
| 2010-09-30 | 3.1.5 | AUTOSAR Administration | • Initial release |

**Disclaimer**

# Contents

# 1 Scope of Document

This document specifies the requirements of the crypto stack:

- [1, Crypto Service Manager (Csm)]
- [2, Crypto Interface (CryIf)]
- [3, Crypto Driver (Crypto)]
- [4, Key Manager]

# 2 Conventions to be used

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([5]).

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([5]).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as follows.

Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the adjective "LEGALLY REQUIRED", means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase, or the phrase "MUST NOT", means that the definition is an absolute prohibition of the specification due to legal issues.
- SHALL: This phrase, or the adjective "REQUIRED", means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

An implementation, which does not include a particular option, SHALL be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, SHALL be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

# 3 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to CryptoStack that are not included in the AUTOSAR Glossary [6].

| *Abbreviation / Acronym:* | *Description:* |
|---|---|
| µC | Microcontroller |
| AES | Advanced Encryption Standard |
| CBC | Cipher Block Chaining |
| CDD | Complex Device Driver |
| CFB | Cipher Feedback |
| CMAC | Cipher-based Message Authentication Code |
| CPU | Central Processing Unit |
| CRYPTO / Crypto | Crypto Driver |
| CRYIF / CryIf | Crypto Interface |
| CSM / Csm | Crypto Service Manager |
| DET / Det | Default Error Tracer |
| ECB | Electronic Code Book |
| ECC | Elliptic Curve Cryptography |
| ECDH | Elliptic Curve Diffie-Hellman |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| ECU | Electronic Control Unit |
| GCM | Galois Counter Mode |
| GMAC | Galois-based Message Authentication Code |
| HMAC | Hash-based Message Authentication Code |
| HSM / Hsm | Hardware Security Module |
| HW | HardWare |
| KEM | Key Encapsulation Mechanism |
| KeyM | Key Manager |
| MAC | Message Authentication Code |
| MCAL | Micro Controller Abstraction Layer |
| OEM | Original Equipment Manufacturer |
| OFB | Output Feedback |
| PKI | Public Key Infrastructure |
| PRNG | Pseudo-Random Number Generator |
| RACE | Rapid Automatic Cryptographic Equipment |
| RAM | Random Access Memory |
| RIPEMD | RACE Integrity Primitives Evaluation Message Digest |
| RSA | Rivest-Shamir-Adleman Cryptosystem |
| RTE | Run Time Environment |
| SHA | Secure Hash Algorithm |
| SECOC / SecOc | Secure Onboard Communication |
| SW | SoftWare |
| SWC | SoftWare Component |
| SWS | SoftWare Specification |
| TRNG | True Random Number Generator |
| Vi | VendorId |
| XEX | Xor-Encrypt-Xor |
| XTS | XEX-based tweaked-codebook mode with ciphertext stealing |

**Table 3.1: Acronyms and abbreviations used in the scope of this Document**

| ***Terms:*** | ***Description:*** | |
|---|---|---|
| Crypto Driver Object | A Crypto Driver Object is an instance of a crypto module (hardware or software), which is able to perform one or more different crypto operations. | |
| User | A user is a configured object with an ID and configured jobs. | |
| Channel | A channel is the path from a Crypto Service Manager queue via the Crypto Interface to a specific Crypto Driver Object. | |
| Job | A job is an instance of a user's configured cryptographic primitive. | |
| Primitive | A primitive is an instance of a configured cryptographic algorithm realized in a Crypto Driver Object. Among others it refers to a functionality provided by the CSM to the application, the concrete underlining 'algorithmfamily' (e.g. AES, MD5, RSA, etc.), and a 'algorithmmode' (e.g. ECB, CBC, etc). | |
| Operation | An operation of a crypto primitive declares what part of the crypto primitive shall be performed. There are three different operations: | |
| | START | Operation indicates a new request of a crypto primitive, and it shall cancel all previous requests. |
| | UPDATE | Operation indicates, that the crypto primitive expect input data. |
| | FINISH | Operation indicates, that after this part all data are fed completely and the crypto primitive can finalize the calculations. |
| | It is also possible to perform more than one operation at once by concatenating the corresponding bits of the operation_mode argument. | |
| Priority | The priority of a user defines the importance of it. The higher the priority (as well in value), the more immediate the user's job will be executed. The priority of a cryptographic job is part of the user's configuration. | |
| Service | A service shall be understand as defined in the TR_Glossary document: A service is a type of operation that has a published specification of interface and behavior, involving a contract between the provider of the capability and the potential clients. | |

**Table 3.2: Terms used in the scope of this Document**

# 4 Requirements Specification

This chapter describes all requirements driving the work to define the CryptoStack.

## 4.1 Functional Overview

The Crypto Stack offers a standardized access to cryptographic services for applications and system functions.

The cryptographic services are, e.g., the computation of hashes, the verification of asymmetrical signatures, or the symmetrical encryption of data. These services depend on underlying cryptographic primitives and cryptographic schemes. The CSM shall make it possible for different applications to use the same service but using different underlying primitives and/or schemes. E.g., one application might need to use the hash service to compute an SHA2 digest and another might need to compute an SHA1 digest. Or one application might need to verify a signature which has been computed with the RSASSA-PKCS1-V1_5 signature scheme and using SHA1 as an underlying hash primitive, while another application might need to verify a signature computed with a different scheme which uses SHA2 as an underlying hash primitive. The Crypto Stack shall make it possible to configure which services are needed and to create several configurations for each service where schemes and primitives can be chosen.

Furthermore, since the computation of many of the cryptographic services is very computation intensive, provisions have to be made for scheduling these long computations. The jobs shall be configurable to be executed synchronously or asynchronously.

The Crypto Stack provides services with cryptography functionality, based on software libraries or on hardware modules. Also, mixed setups are possible, for example if a hardware module cannot supply the necessary functionality on its own. In the following, we refer to all instantiations of underlying functionality, be it hardware or software, as "crypto library".

### 4.1.1 Supported Algorithms

The following cryptographic algorithms or primitives should be supported by the Crypto Stack:

- Random Number Generation
  - Deterministic Random Number Generator (DRNG)
  - True Random Number Generator (TRNG)
- Symmetric Encryption
  - AES

    - Key Length: 128 and 256 bits
    - Modes: ECB, CBC, CTR, GCM, OFB, CFB, XTS
  - PRESENT
    - Key Length: 128 bits
    - Modes: ECB, CBC, CTR, GCM, OFB, CFB, XTS
  - ChaCha12/ChaCha20
    - Key Length: 256 bits
- AEAD
  - AES GCM
  - ChaCha20 and POLY1305
- Asymmetric Encryption/Decryption and Signature Handling
  - RSA
    - Key Length: 1024, 2048, 3072, 4096
    - Padding: PKCS#1 v2.2
  - ECDSA
    - Key Length: 256 bits
    - Curve: NIST P-256
  - Ed25519
- Hash
  - SHA-2
    - Length: 224, 256, 384, 512
  - SHA-3
    - Length: 224, 256, 384, 512
  - BLAKE
    - Length: 224, 256, 384, 512
  - RIPEMD-160
- MAC
  - CMAC
  - GMAC
  - HMAC

  - POLY1305
- Key exchange:
  - ECDH
    - Key Length: 256 bits
    - Curve: NIST P-256
  - X25519

## 4.2 Functional Requirements

### 4.2.1 Crypto Stack

#### 4.2.1.1 General

**[SRS_CryptoStack_00100] Synchronous Job Processing**

*Upstream requirements:* RS_BRF_01456

⌈

| | |
|---|---|
| ***Description:*** | Some crypto services shall allow synchronous job processing. |
| ***Rationale:*** | There are some crypto services which can be calculated very fast and are required very fast. Then, the overhead of the asynchronous job processing including main function calls and call back functions, is too big. |
| ***Use Case:*** | MAC generation for the SecOC module |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00101] Asynchronous Job Processing**

*Upstream requirements:* RS_BRF_01456

⌈

| | |
|---|---|
| ***Description:*** | Some crypto services shall allow asynchronous job processing. |
| ***Rationale:*** | There are some crypto services which require a lot of time or are executed in an HSM. Then, synchronous job processing would require too much time. |
| ***Use Case:*** | Signature verification |
| ***Dependencies:*** | – |

▽

△

| | |
|---|---|
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00003] The crypto stack shall be able to incorporate modules of the crypto library**

*Upstream requirements:* RS_BRF_02032

⌈

| | |
|---|---|
| ***Description:*** | The crypto stack shall be able to incorporate modules of a crypto library. |
| ***Rationale:*** | The crypto library itself has to be available in the AUTOSAR stack. |
| ***Use Case:*** | SW implementation of cryptographic primitives. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.1.2 Configuration

**[SRS_CryptoStack_00007] The Crypto Stack shall provide scalability for the cryptographic features**

*Upstream requirements:* RS_BRF_01456, RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall guarantee that the unused cryptographic features are not compiled into the binary. |
| ***Rationale:*** | Different security features require different encryption solutions (example: symmetric/asymmetric encryption, hashing) with or without hardware support. The hardware profiles available offer different features (example: internal NVM, random number generator, secure CPU core...). Scalability of cryptographic features allow different strategies for implementation if some features are not required and thus minimize SW or HW resource utilization. |
| ***Use Case:*** | The mapping between crypto stack and the functionalities of microcontroller hardware allows hardware vendors to develop generic drivers for their HSMs. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00008] The Crypto Stack shall allow static configuration of keys used for cryptographic jobs

*Upstream requirements:* RS_BRF_02031, RS_BRF_01946

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall allow static configuration of symmetric and asymmetric key pairs used for crypto services. |
| ***Rationale:*** | It shall be possible to use keys individually. |
| ***Use Case:*** | Data encryption with a protected key in the HSM. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00105] The Crypto Stack shall only allow unique key identifiers

*Upstream requirements:* RS_BRF_02031, RS_BRF_01946

⌈

| | |
|---|---|
| ***Description:*** | There is one keyId configured for each cryptographic key. |
| ***Rationale:*** | It shall be possible to treat keys individually. |
| ***Use Case:*** | Usage of cryptographic keys. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00013] The modules of the crypto stack shall support only pre-compile time configuration

*Upstream requirements:* RS_BRF_01136

⌈

| | |
|---|---|
| ***Description:*** | The modules of the crypto stack shall support only pre-compile time configuration. |
| ***Rationale:*** | No applicable post-build or link-time parameters |
| ***Use Case:*** | All the configurable parameter values must be decided before compile or build time. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00094] The configuration files of the crypto stack modules shall be readable for human beings**

*Upstream requirements:* RS_BRF_01456

⌈

| | |
|---|---|
| ***Description:*** | The configuration files of the crypto stack modules shall be readable for human beings:<br>e.g. by integration of comments or by tool - support. |
| ***Rationale:*** | Human being have to read and understand the configuration. So the configuration shall be readable and understandable for human being. |
| ***Use Case:*** | Debugging |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.1.3 Initialization

None

#### 4.2.1.4 Normal Operation

**[SRS_CryptoStack_00009] The Crypto Stack shall support reentrancy for all crypto services**

*Upstream requirements:* RS_BRF_02033

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall support reentrancy of crypto related interfaces to enable parallel operations of the same or different type when requested by multiple users.<br>This requirement also covers scenarios where applications are residing on different cores. |
| ***Rationale:*** | Crypto jobs shall be processable simultaneously |
| ***Use Case:*** | Different applications may use cryptographic services in parallel. Handling of different tasks at the same time is necessary. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00010] The Crypto Stack shall conceal symmetric keys from the users of crypto services

*Upstream requirements:* RS_BRF_02031, RS_BRF_01946

⌈

| | |
|---|---|
| ***Description:*** | There shall be no interface to extract symmetric key values directly to the user. Keys shall be addressed via identifiers by the users.<br>Such keys shall only be exported in an encrypted format. |
| ***Rationale:*** | If keys are stored in the application, this increases the chances of invalidation of keys or keys being compromised. |
| ***Use Case:*** | Keys residing in the HSM |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00011] The Crypto Stack shall conceal asymmetric private keys from the users of Crypto services

*Upstream requirements:* RS_BRF_02031, RS_BRF_01946

⌈

| | |
|---|---|
| ***Description:*** | There shall be no interface to extract asymmetric private key values directly to the user. Keys shall be addressed via identifiers by the Users.<br>Such keys shall only be exported in an encrypted format. |
| ***Rationale:*** | If keys are stored in the application, this increases the chances of invalidation of keys or keys being compromised. |
| ***Use Case:*** | Keys residing in the HSM |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00019] The Crypto Stack shall identify random number generation as a cryptographic primitive which can be requested to a driver

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall identify random number generation as a cryptographic primitive which can be requested to a driver. |
| ***Rationale:*** | Random number Generators residing on different crypto drivers should be accessed using a homogenous interface. |

▽

△

| | |
|---|---|
| ***Use Case:*** | Generate random number |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00020] The Crypto Stack shall identify symmetric encryption/decryption as a cryptographic primitive which can be requested to a driver

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall identify symmetric encryption/decryption as a cryptographic primitive which can be requested to a driver. |
| ***Rationale:*** | Symmetric algorithms residing on different crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | Encrypted communication |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00021] The Crypto Stack shall identify asymmetric encryption/decryption as a cryptographic primitive which can be requested to a driver

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall identify asymmetric encryption/decryption as a cryptographic primitive which can be requested to a driver. |
| ***Rationale:*** | Asymmetric algorithms residing on different crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | Unique Interface for success of heterogeneous hardware- and software-solutions |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00022] The Crypto Stack shall identify MAC generation/verification as a cryptographic primitive which can be requested to a driver

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall identify MAC generation/verification as a cryptographic primitive which can be requested to a driver. |
| ***Rationale:*** | MAC algorithms residing on different crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | SecOC using MACs to verify messages |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00023] The Crypto Stack shall identify asymmetric signature generation/verification as a cryptographic primitive which can be requested to a driver

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall identify asymmetric signature generation/verification as a cryptographic primitive which can be requested to a driver. |
| ***Rationale:*** | Asymmetric signature algorithms residing on different crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | Signature creation/verification |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00024] The Crypto Stack shall identify hash calculation as a cryptographic primitive which can be requested to a driver

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall identify hash calculation as a cryptographic primitive which can be requested to a driver. |
| ***Rationale:*** | Hash algorithms residing on different crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | Signature verification |

▽

△

| | |
|---|---|
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00026] The Crypto Stack shall provide an interface for the generation of asymmetric keys

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall provide an abstracted interface for the generation of asymmetric key pair service. |
| ***Rationale:*** | Key generation services residing on different Crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | Generation of an asymmetric key pair inside the ECU. Then, the private key never has to be available outside the ECU. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00027] The Crypto Stack shall provide an interface for the generation of symmetric keys

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall abstract the user from multiple symmetric keys stored by various Crypto Drivers through a standardized interface. Also, it shall provide an interface to the driver for generation of such keys. |
| ***Rationale:*** | Key generation services residing on different Crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | Password-based key input |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00103] The Crypto Stack shall provide an interface for the derivation of symmetric keys

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall abstract the user from multiple symmetric keys stored by various Crypto Drivers through a standardized interface. Also, it shall provide an interface to the driver for derivation of such keys. |
| ***Rationale:*** | Key derivation services residing on different Crypto drivers should be accessed using a homogenous interface. |
| ***Use Case:*** | Password-based key input |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00028] The Crypto Stack shall provide an interface for key exchange mechanisms

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall support key exchange mechanism as a key management interface |
| ***Rationale:*** | Key exchange algorithms residing on different crypto drivers should be accessed using a homogenous interface |
| ***Use Case:*** | Session handling |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00029] The Crypto Stack shall provide an interface for key wrapping/extraction mechanisms

*Upstream requirements:* RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall support key wrapping (encapsulation) and extraction mechanism forward such requests from CSM to the respective driver. It shall support wrapping using a symmetric key as well as asymmetric key. |
| ***Rationale:*** | Key wrapping and encapsulation algorithms implemented in the driver shall not be accessed directly by the users and need to be abstracted. |

▽

△

| | |
|---|---|
| ***Use Case:*** | Session handling |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00031] The Crypto Stack shall provide an interface for parsing certificates

*Upstream requirements:* RS_BRF_01946

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Stack shall support parsing certificates and extracting the contained keys |
| ***Rationale:*** | The crypto driver shall parse incoming certificates and store the key information in the corresponding key |
| ***Use Case:*** | For PKI it is necessary to obtain public keys out of certificates |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00061] The Crypto Stack shall support detection of invalid keys

*Upstream requirements:* RS_BRF_02031, RS_BRF_01946

⌈

| | |
|---|---|
| ***Description:*** | The implementation of a cryptographic primitive shall detect and reject invalid keys. |
| ***Rationale:*** | Algorithms like RSA or several ECC flavors know keys which can be used to perform the mathematical foundation of the algorithm without an error but address special corner cases and are not secure to handle. Keys like that have to be identified and rejected. There is no generic approach hence the implementation has to be in the cryptographic primitive itself or, in case hardware is used, in its driver. |
| ***Use Case:*** | RSA and several Elliptic Curve Cryptosystems |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.1.5 Shutdown Operation

None

#### 4.2.1.6 Fault Operation

None

### 4.2.2 Key Manager

#### 4.2.2.1 General

**[SRS_CryptoStack_00106] Key manager operation shall either run synchronously or asynchronously.**

*Upstream requirements:* RS_Main_00001

⌈

| | |
|---|---|
| ***Description:*** | A key manager operation shall support synchronous mode or asynchronous mode, if the computation takes a long time. . For the latter case, a callback shall indicate that an operation has been completed. |
| ***Rationale:*** | To avoid multiple task activation a key management operation shall be executed in the background. |
| ***Use Case:*** | Key derivation can take considerable amount of time |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

**[SRS_CryptoStack_00107] Key manager shall provide interfaces to generate or update key material.**

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The KeyM module shall provide an API allowing a software component or DCM to generate or update the key material for an ECU. KeyM shall use the CSM for the required cryptographic operations and key storage |
| ***Rationale:*** | A basic software module e.g. DCM or software component) triggers to generate or re-generate key material. The KeyM generates the key material and stores it in the crypto driver by calling the appropriate functions of the CSM. |
| ***Use Case:*** | Key server wants to provide key material |

▽

△

| | |
|---|---|
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00108] Key manager shall be able to negotiate a shared secret by exchanging messages with other ECUs

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The KeyM module shall support key negotiation between ECUs via interfaces to the PduR. |
| ***Rationale:*** | Key management tasks such as multi-party key negotiation can require the exchange of messages to agree on keys. |
| ***Use Case:*** | Onboard negotiation of keys |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00109] Key manager shall be able to manage derivation of key material from a common secret

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The KeyM module shall provide an API for deriving keys from a common shared secret. |
| ***Rationale:*** | Secret keys (e.g. for secure onboard communication) have to be shared between multiple ECUs. One way to handle this is to derive the keys from a secret which is shared between the participating ECUs. |
| ***Use Case:*** | The key server provides a common secret and the key manager derives several symmetric keys out of this common secret. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00110] The KeyM module shall support on-board generated keys

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The KeyM shall provide an API for handling of on-board generated keys. |
| ***Rationale:*** | On-board generated keys have to be negotiated between two or more ECUs. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00111] The KeyM module shall support verification of certificates based on configured rules

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The certificate verification operations provided by KeyM shall allow to check the values of configured certificate elements as part of the verification operation. |
| ***Rationale:*** | Besides the verification of signatures and basic certificate elements, certificate verification may require checks whether custom certificate elements conform to specific rules. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00112] The KeyM module shall support retrieving arbitrary elements of a certificate

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | KeyM shall provide operations to retrieve the value of arbitrary certificate elements |
| ***Rationale:*** | Applications and basic software modules may need to use the value of specific certificate elements to perform operations. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |

▽

△

| | |
|---|---|
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

**[SRS_CryptoStack_00125] AUTOSAR certificate handler shall support the certificate formats CVC and X.509.**

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | AUTOSAR certificate handler shall support the certificate formats CVC and X.509. |
| ***Rationale:*** | Only a subset of common known and used client certificate types is supported by AUTOSAR. This allows a standardized handling and evaluation of certificates and content. |
| ***Use Case:*** | Certificates issued by OEM are supported by the ECU. E.g. The OEM PKI issues a certificate for a repair shop diagnostic tester. The diagnostic tester authenticates itself with the ECU. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

#### 4.2.2.2 Configuration

**[SRS_CryptoStack_00113] Keys in the crypto stack can be uniquely identified**

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | All keys maintained by KeyM must be uniquely addressed so that new key material provided by the key server can be assigned to the corresponding key in the crypto driver. |
| ***Rationale:*** | A symbolic identification is required so that KeyM can uniquely identify which key is used by a corresponding job. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00114] Crypto driver shall place keys into specific key slots

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | A key shall be placed to a specific key slot. This is required for the SHE to provide the correct key update information. |
| ***Rationale:*** | The SHE update protocol information includes the slot of the key that shall be updated. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00115] KeyM shall be highly configurable to support different OEM use cases

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The key management provides various options like certificate handling, key management operation. Also, there is already a various number of key management systems established that must be fulfilled with this module. This requires a high flexibility of the module to adapt it to different use cases. |
| ***Rationale:*** | The Key Management depends highly on OEM specific processes and formats. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

#### 4.2.2.3 Initialization

### [SRS_CryptoStack_00116] Keys shall use a default value if configured

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | It shall be possible to configure KeyM to use key default values. |
| ***Rationale:*** | A key server may not be available during development of a secure system. So it should be possible to configure default keys which can be used during development. |
| ***Use Case:*** | System tests during development phase. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### [SRS_CryptoStack_00117] Keys shall not be used if they are empty or corrupted

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | A key that has no default value or has been programmed but its contents cannot be retrieved shall be marked as invalid and is therefore not usable. |
| ***Rationale:*** | A system that has no properly initialized keys shall not provide any undetermined cryptographic results. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

#### 4.2.2.4 Normal Operation

**[SRS_CryptoStack_00118] Key material shall be securely stored either in NVM or CSM**

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The KeyM shall use CSM interfaces for securely storing keys. |
| ***Rationale:*** | Off-board generated keys and on-board generated keys have to be securely stored. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

**[SRS_CryptoStack_00119] Provide a proof that the key has been programmed correctly**

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The KeyM shall provide interfaces for providing a proof for correctly programmed keys.<br>A verification function can be used to provide a proof to the key server if the correct key is programmed and associated with a specific job. |
| ***Rationale:*** | Verification of a succesful key installation. A key provider may want to check if the key has been programmed correctly. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

#### 4.2.2.5 Shutdown Operation

### [SRS_CryptoStack_00120] Cleanup all key material on shutdown operation

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | Keys may be temporarily stored in RAM. A shutdown operation may cleanup these keys by removing the key data from RAM. |
| ***Rationale:*** | Keys may be stored during shutdown operation.<br>For security reason, keys in RAM shall be destroyed. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

#### 4.2.2.6 Fault Operation

### [SRS_CryptoStack_00121] Pass security related events to security event memory (SEM) for secure logging

*Status:* DRAFT

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The KeyM shall use the interfaces provided by the security event memory for secure logging of security related events which are produced by KeyM. |
| ***Rationale:*** | Events such as failed key negotiation are security related events which need to be logged. |
| ***Use Case:*** | C2 - Key Management |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C2 |

⌋

### 4.2.3 Crypto Service Manager

#### 4.2.3.1 General

**[SRS_CryptoStack_00006] Each primitive of the CRYIF shall belong to exactly one service of the CSM**

*Upstream requirements:* RS_BRF_02032

⌈

| | |
|---|---|
| ***Description:*** | Each primitive of the CRYIF shall belong to exactly one service of the CSM. |
| ***Rationale:*** | There are channels which map a user specific crypto primitive via the CRYIF to the underlying Crypto Driver module. |
| ***Use Case:*** | The CRYIF is responsible for each service to map to the corresponding Crypto Driver module |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.3.2 Configuration

**[SRS_CryptoStack_00079] The job processing mode (synchronous or asynchronous) of a CSM service shall be defined by static configuration**

*Upstream requirements:* RS_BRF_01456, RS_BRF_01136

⌈

| | |
|---|---|
| ***Description:*** | The mode of cryptographic jobs provided by the CSM shall be defined by static configuration. |
| ***Rationale:*** | It shall not be possible to change the behavior of a specific CSM service during runtime. |
| ***Use Case:*** | Synchronous hash calculation |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00102] The priority of a user and its crypto jobs shall be defined by static configuration** ⌈

| | |
|---|---|
| ***Description:*** | The user's priority shall be defined by static configuration. All jobs of that user inherit that priority. |
| ***Rationale:*** | There are crypto jobs which have to processed very fast (e.g. MAC generation for the SecOC). Other crypto jobs (e.g. hashing over the whole ROM) take very long but are not time critical. |
| ***Use Case:*** | Prioritized job processing |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00080] The set of cryptographic services provided by the CSM shall be defined by static configuration**

*Upstream requirements:* RS_BRF_01456, RS_BRF_01136

⌈

| | |
|---|---|
| ***Description:*** | The set of cryptographic services provided by the CSM shall be defined by static configuration. |
| ***Rationale:*** | It is not possible during runtime to add new CSM services. |
| ***Use Case:*** | If symmetrical encryption is supported by the driver, it has to be configured which user with which is key is using it. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00081] The CSM module specification shall specify which other modules are required**

*Upstream requirements:* RS_BRF_01456, RS_BRF_01064

⌈

| | |
|---|---|
| ***Description:*** | The CSM module specification shall specify which other modules are required. |
| ***Rationale:*** | Basic functionality |
| ***Use Case:*** | – |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00082] The CSM module specification shall specify the interface and behavior of the callback function, if the asynchronous job processing mode is selected**

*Upstream requirements:* RS_BRF_01456, RS_BRF_01064

⌈

| | |
|---|---|
| ***Description:*** | The CSM module specification shall specify how the callback function has to be implemented, if the asynchronous job processing mode is selected. |
| ***Rationale:*** | The CSM has to call the callback function. Thus, the CSM has to know the signature of the callback function. |
| ***Use Case:*** | – |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.3.3 Initialization

#### 4.2.3.4 Normal Operation

**[SRS_CryptoStack_00084] The CSM module shall use the streaming approach for some selected services**

*Upstream requirements:* RS_BRF_01456

⌈

| | |
|---|---|
| ***Description:*** | The CSM module shall use the streaming approach for some provided services (see Software Specification of CSM). |
| ***Rationale:*** | Basic functionality |
| ***Use Case:*** | It shall be possible to hand over the input data in small chunks to the service. |
| ***Dependencies:*** | – |

⌋

#### 4.2.3.5 Shutdown Operation

<what to do when the module is shut down>

#### 4.2.3.6 Fault Operation

**[SRS_CryptoStack_00086] The CSM module shall distinguish between error types**

*Upstream requirements:* RS_BRF_02168, RS_BRF_02272

⌈

| | |
|---|---|
| ***Description:*** | The CSM module shall distinguish between the following two types or errors:<br>- errors that can only occur during development<br>- errors that are expected to occur also in production code |
| ***Rationale:*** | Basic functionality |
| ***Use Case:*** | – |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00087] The CSM module shall report detected development errors to the Default Error Tracer**

*Upstream requirements:* RS_BRF_02168, RS_BRF_02272

⌈

| | |
|---|---|
| ***Description:*** | The CSM module shall report detected development errors to the Default Error Tracer |
| ***Rationale:*** | Basic functionality |
| ***Use Case:*** | – |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00096] The CSM module shall not return specific development error codes via the API**

*Upstream requirements:* RS_BRF_00129, RS_BRF_02168

⌈

| | |
|---|---|
| ***Description:*** | The CSM module shall not return specific development error codes via the API. In case of a detected development error the error shall only be reported to the DET. If the API function which detected the error has the return type Std_ ReturnType, it shall return E_NOT_OK. |

▽

△

| | |
|---|---|
| ***Rationale:*** | Basic functionality |
| ***Use Case:*** | – |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00097] The CSM shall check passed API parameters for validity

*Upstream requirements:* RS_BRF_00129, RS_BRF_02168, RS_BRF_02232

⌈

| | |
|---|---|
| ***Description:*** | The CSM shall check passed API parameters for validity. This checking shall be statically configurable for those errors that only can occur during development. |
| ***Rationale:*** | Basic functionality |
| ***Use Case:*** | Debugging |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

## 4.2.4 Crypto Interface

### 4.2.4.1 Configuration

### [SRS_CryptoStack_00014] The Crypto Interface shall have an interface to the static configuration information of the Crypto Driver

*Upstream requirements:* RS_BRF_01008

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Interface shall have an interface to the static configuration information of the Crypto Driver. |
| ***Rationale:*** | Flexibility and scalability |
| ***Use Case:*** | To derive vendor API Infix to support multiple Crypto Drivers |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00015] Channels mapped to different Crypto Driver Objects shall be uniquely configurable in Crypto Interface**

*Upstream requirements:* RS_BRF_02032, RS_BRF_01456, RS_BRF_01136

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Interface shall support a configuration model where all virtual channels shall be statically mapped to Crypto Driver Objects.<br>Virtual channels are the virtual way from the queue of the CSM over the CRYIF to the corresponding Crypto Driver Object. |
| ***Rationale:*** | Each crypto driver object in the driver can be abstracted from and utilized by CSM using virtual channels. |
| ***Use Case:*** | Two hardware resources: one for symmetric cryptography, one for asymmetric cryptography |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.4.2 Initialization

#### 4.2.4.3 Normal Operation

#### 4.2.4.4 Shutdown Operation

None

#### 4.2.4.5 Fault Operation

**[SRS_CryptoStack_00034] The Crypto Interface shall report detected development errors to the Default Error Tracer**

*Upstream requirements:* RS_BRF_02232

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Interface shall report detected development errors to the Default Error Tracer (DET).<br>The detection and reporting shall be statically configurable with one single preprocessor switch. |
| ***Rationale:*** | Debugging Support |

▽

△

| | |
|---|---|
| ***Use Case:*** | All the input parameters and internal states have to be validated before processing. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### 4.2.5 Crypto Driver

#### 4.2.5.1 Configuration

**[SRS_CryptoStack_00036] The Crypto Driver shall allow static configuration of Crypto Driver Objects**

*Upstream requirements:* RS_BRF_01456, RS_BRF_01136

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Driver shall allow defining of different Crypto Driver Objects. |
| ***Rationale:*** | Abstraction of different hardware units |
| ***Use Case:*** | Parallel processing of symmetric operations and asymmetric operations |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00104] Crypto Interface keys mapped to different Crypto Driver Keys shall be uniquely configurable in the Crypto Interface**

*Upstream requirements:* RS_BRF_02032, RS_BRF_01456, RS_BRF_01136

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Interface shall support a configuration model where all CRYIF keys shall be statically mapped to keys in the crypto driver. |
| ***Rationale:*** | Similar to the channels where the CsmQueue is mapped to the Crypto Driver Object, the keys in the CSM have to mapped via the CRYIF to the corresponding key in the Crypto Driver. |
| ***Use Case:*** | Multiple Crypto Driver modules where each module has its own key identifiers |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.5.2 Initialization

#### 4.2.5.3 Normal Operation

**[SRS_CryptoStack_00098] The Crypto Driver shall provide access to all cryptographic algorithms supported by the hardware** ⌈

| | |
|---|---|
| ***Description:*** | The Crypto Driver shall support access to all by the Crypto Stack supported algorithms. |
| ***Rationale:*** | Usage of hardware support and performance benefits. |
| ***Use Case:*** | Primitives which are supported by the HSM should be accessible through the Crypto Driver |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

#### 4.2.5.4 Shutdown Operation

None

#### 4.2.5.5 Fault Operation

### 4.2.6 Security Event Memory

#### 4.2.6.1 General

**[SRS_CryptoStack_00122] Log security events reported by basic software modules and SWC**

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The security event memory (SEM) shall enable to log security events (SE) which are monitored by BSW modules or SWCs.<br>It shall be possible to store the type of the security event along with data which is useful for forensic analysis. |
| ***Rationale:*** | Log security events in a way that is useful for forensic analysis |
| ***Use Case:*** | UC1.1: BSW modules or SWCs write security related events to the SEM |

▽

△

| | |
|---|---|
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C1 |

⌋

### [SRS_CryptoStack_00123] Configure security event properties

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | Depending on the type of security event the number of storable snap shots and the properties of the snap shots shall be configurable. |
| ***Rationale:*** | Depending on the type of event different data needs to be stored in the snapshot for the event. Depending on the expected frequency of the event specific numbers of storable snapshots are sufficient. |
| ***Use Case:*** | UC1.1: BSW modules or SWCs write security related events to the SEM |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C1 |

⌋

### [SRS_CryptoStack_00124] Allow authorized users to read SEM data via diagnostic interfaces

*Upstream requirements:* RS_Main_00514

⌈

| | |
|---|---|
| ***Description:*** | The SEM shall provide SEM data via diagnostic interfaces.<br>SEM data shall be protected from unauthorized read and write access via diagnostic interfaces. |
| ***Rationale:*** | Diagnostic testers are used to access SEM data |
| ***Use Case:*** | UC1.2: Authorized stakeholders read data from the SEM for off-board forensic analysis |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | Concept 636 "Security Extensions" - C1 |

⌋

## 4.3 Non-Functional Requirements (Qualities)

### 4.3.1 General

### 4.3.2 Crypto Service Manager

**[SRS_CryptoStack_00088] The CSM module shall provide an abstraction layer which offers a standardized interface to higher software layers to access cryptographic algorithms**

*Upstream requirements:* RS_BRF_01456, RS_BRF_01016, RS_BRF_01056

⌈

| | |
|---|---|
| ***Description:*** | The CSM module shall provide an abstraction layer which offers a standardized interface to higher software layers to access cryptographic algorithms. |
| ***Rationale:*** | An abstraction layer encapsulates internal behaviors and reduces complexity. It also increases maintainability, improves portability and eases testability. |
| ***Use Case:*** | Session handling. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

**[SRS_CryptoStack_00089] The CSM module shall be located in the AUTOSAR service layer**

*Upstream requirements:* RS_BRF_01016, RS_BRF_01408

⌈

| | |
|---|---|
| ***Description:*** | The CSM module shall be located in the Autosar service layer |
| ***Rationale:*** | Management functionality must be available to all modules and layers of the system |
| ***Use Case:*** | The CSM module shall be accessible from applications above the RTE. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00090] The CSM shall provide an interface to be accessible via the RTE

*Upstream requirements:* RS_BRF_01408, RS_BRF_01280

⌈

| | |
|---|---|
| ***Description:*** | The CSM shall provide an interface to be accessible via the RTE. |
| ***Rationale:*** | The CSM module shall be accessible from applications above the RTE. |
| ***Use Case:*** | Applications which require crypto services. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00091] The CSM shall provide one Provide–Port for each configuration

*Upstream requirements:* RS_BRF_01056, RS_BRF_01280, RS_BRF_01408, RS_BRF_01456

⌈

| | |
|---|---|
| ***Description:*** | The CSM shall provide one Provide–Port for each configuration. All configured services shall be accessible via this port. |
| ***Rationale:*** | All crypto services shall be accessible from applications above the RTE via the Provide-Port. |
| ***Use Case:*** | All applications request to the CSM uses this port. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### [SRS_CryptoStack_00092] The CSM shall provide one Require-Port for each configuration

*Upstream requirements:* RS_BRF_01056, RS_BRF_01280, RS_BRF_01408, RS_BRF_01456

⌈

| | |
|---|---|
| ***Description:*** | The CSM shall provide one Require-Port for each configuration. The configured callback function shall be accessible via this port. |
| ***Rationale:*** | All crypto services shall have access to the configured callback functions via the Require-Port. |
| ***Use Case:*** | Most asynchronous services own a callback function. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### 4.3.3 Crypto Interface

**[SRS_CryptoStack_00075] The Crypto Interface shall be the interface layer between the underlying crypto driver(s) and upper layers**

*Upstream requirements:* RS_BRF_01000, RS_BRF_01008, RS_BRF_01016

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Interface is the single interface for all upper Layer (BSW) for crypto operations.<br>The Crypto Interface is the single user of the Crypto Driver |
| ***Rationale:*** | Interfaces and interaction |
| ***Use Case:*** | Different Users might need to access more than one Crypto Drivers or SW based solutions. Also, different upper layers might need to access crypto services. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | AUTOSAR_WP Architecture_SoftwareArchitecture |

⌋

**[SRS_CryptoStack_00076] The Crypto Interface implementation and interface shall be independent from underlying Crypto Hardware or Software**

*Upstream requirements:* RS_BRF_01000, RS_BRF_02031

⌈

| | |
|---|---|
| ***Description:*** | The CRYIF implementation and CRYIF interfaces shall be independent from the underlying Crypto Driver modules. |
| ***Rationale:*** | Portability and reusability |
| ***Use Case:*** | Encapsulate implementation details of a specific Crypto module from higher software layers. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

### 4.3.4 Crypto Driver

**[SRS_CryptoStack_00095] The Crypto Driver module shall strictly separate error and status information**

*Upstream requirements:* RS_BRF_00129, RS_BRF_02168, RS_BRF_02232, RS_BRF_02272

⌈

| | |
|---|---|
| ***Description:*** | The Crypto Driver module shall strictly separate error and status information. This requirement applies to return values and also to internal variables. |
| ***Rationale:*** | The distinction between error and status information allow an easier handling with them. Errors shall be treated differently than status information. |
| ***Use Case:*** | All return values are error information. |
| ***Dependencies:*** | – |
| ***Supporting Material:*** | – |

⌋

# 5 Requirements Tracing

The following table references the features specified in [7] and links to the fulfillments of these.

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_BRF_00129]** | AUTOSAR shall support data corruption detection and protection | [SRS_CryptoStack_00095]<br>[SRS_CryptoStack_00096]<br>[SRS_CryptoStack_00097] |
| **[RS_BRF_01000]** | AUTOSAR architecture shall organize the BSW in a hardware independent and a hardware dependent layer | [SRS_CryptoStack_00075]<br>[SRS_CryptoStack_00076] |
| **[RS_BRF_01008]** | AUTOSAR shall organize the hardware dependent layer in a microcontroller independent and a microcontroller dependent layer | [SRS_CryptoStack_00014]<br>[SRS_CryptoStack_00075] |
| **[RS_BRF_01016]** | AUTOSAR shall provide a modular design inside software layers | [SRS_CryptoStack_00075]<br>[SRS_CryptoStack_00088]<br>[SRS_CryptoStack_00089] |
| **[RS_BRF_01056]** | AUTOSAR BSW modules shall provide standardized interfaces | [SRS_CryptoStack_00088]<br>[SRS_CryptoStack_00091]<br>[SRS_CryptoStack_00092] |
| **[RS_BRF_01064]** | AUTOSAR BSW shall provide callback functions in order to access upper layer modules | [SRS_CryptoStack_00081]<br>[SRS_CryptoStack_00082] |
| **[RS_BRF_01136]** | AUTOSAR shall support variants of configured BSW data resolved after system start-up | [SRS_CryptoStack_00013]<br>[SRS_CryptoStack_00015]<br>[SRS_CryptoStack_00036]<br>[SRS_CryptoStack_00079]<br>[SRS_CryptoStack_00080]<br>[SRS_CryptoStack_00104] |
| **[RS_BRF_01280]** | AUTOSAR RTE shall offer the external interfaces between Software Components and between Software Components and BSW | [SRS_CryptoStack_00090]<br>[SRS_CryptoStack_00091]<br>[SRS_CryptoStack_00092] |
| **[RS_BRF_01408]** | AUTOSAR shall provide a service layer that is accessible from each basic software layer | [SRS_CryptoStack_00089]<br>[SRS_CryptoStack_00090]<br>[SRS_CryptoStack_00091]<br>[SRS_CryptoStack_00092] |
| **[RS_BRF_01456]** | AUTOSAR services shall provide system wide cryptographic functionality | [SRS_CryptoStack_00007]<br>[SRS_CryptoStack_00015]<br>[SRS_CryptoStack_00036]<br>[SRS_CryptoStack_00079]<br>[SRS_CryptoStack_00080]<br>[SRS_CryptoStack_00081]<br>[SRS_CryptoStack_00082]<br>[SRS_CryptoStack_00084]<br>[SRS_CryptoStack_00088]<br>[SRS_CryptoStack_00091]<br>[SRS_CryptoStack_00092]<br>[SRS_CryptoStack_00094]<br>[SRS_CryptoStack_00100]<br>[SRS_CryptoStack_00101]<br>[SRS_CryptoStack_00104] |

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_BRF_01946]** | AUTOSAR microcontroller abstraction shall provide access to cryptographic hardware | [SRS_CryptoStack_00008] [SRS_CryptoStack_00010] [SRS_CryptoStack_00011] [SRS_CryptoStack_00031] [SRS_CryptoStack_00061] [SRS_CryptoStack_00105] |
| **[RS_BRF_02031]** | AUTOSAR shall provide uniform access to cryptographic solutions implemented either by software or hardware | [SRS_CryptoStack_00007] [SRS_CryptoStack_00008] [SRS_CryptoStack_00010] [SRS_CryptoStack_00011] [SRS_CryptoStack_00019] [SRS_CryptoStack_00020] [SRS_CryptoStack_00021] [SRS_CryptoStack_00022] [SRS_CryptoStack_00023] [SRS_CryptoStack_00024] [SRS_CryptoStack_00026] [SRS_CryptoStack_00027] [SRS_CryptoStack_00028] [SRS_CryptoStack_00029] [SRS_CryptoStack_00061] [SRS_CryptoStack_00076] [SRS_CryptoStack_00103] [SRS_CryptoStack_00105] |
| **[RS_BRF_02032]** | AUTOSAR security shall allow integration of cryptographic primitives into the cryptographic service manager | [SRS_CryptoStack_00003] [SRS_CryptoStack_00006] [SRS_CryptoStack_00015] [SRS_CryptoStack_00104] |
| **[RS_BRF_02033]** | AUTOSAR shall provide concurrent access to cryptographic services | [SRS_CryptoStack_00009] |
| **[RS_BRF_02168]** | AUTOSAR diagnostics shall provide a central classification and handling of abnormal operative conditions | [SRS_CryptoStack_00086] [SRS_CryptoStack_00087] [SRS_CryptoStack_00095] [SRS_CryptoStack_00096] [SRS_CryptoStack_00097] |
| **[RS_BRF_02232]** | AUTOSAR shall support development with run-time assertion checks | [SRS_CryptoStack_00034] [SRS_CryptoStack_00095] [SRS_CryptoStack_00097] |
| **[RS_BRF_02272]** | AUTOSAR shall offer tracing of application software behavior | [SRS_CryptoStack_00086] [SRS_CryptoStack_00087] [SRS_CryptoStack_00095] |
| **[RS_Main_00001]** | Real-Time System Software Platform | [SRS_CryptoStack_00106] |

▽

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_Main_00514]** | System Security Support | [SRS_CryptoStack_00107]<br>[SRS_CryptoStack_00108]<br>[SRS_CryptoStack_00109]<br>[SRS_CryptoStack_00110]<br>[SRS_CryptoStack_00111]<br>[SRS_CryptoStack_00112]<br>[SRS_CryptoStack_00113]<br>[SRS_CryptoStack_00114]<br>[SRS_CryptoStack_00115]<br>[SRS_CryptoStack_00116]<br>[SRS_CryptoStack_00117]<br>[SRS_CryptoStack_00118]<br>[SRS_CryptoStack_00119]<br>[SRS_CryptoStack_00120]<br>[SRS_CryptoStack_00121]<br>[SRS_CryptoStack_00122]<br>[SRS_CryptoStack_00123]<br>[SRS_CryptoStack_00124]<br>[SRS_CryptoStack_00125] |

**Table 5.1: Requirements Tracing**

# 6 References

[1] Specification of Crypto Service Manager
AUTOSAR_CP_SWS_CryptoServiceManager

[2] Specification of Crypto Interface
AUTOSAR_CP_SWS_CryptoInterface

[3] Specification of Crypto Driver
AUTOSAR_CP_SWS_CryptoDriver

[4] Specification of Key Manager
AUTOSAR_CP_SWS_KeyManager

[5] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[6] Glossary
AUTOSAR_FO_TR_Glossary

[7] Requirements on AUTOSAR Features
AUTOSAR_CP_RS_Features