| Document Title | Specification of Cryptography |
|---|---|
| **Document Owner** | AUTOSAR |
| **Document Responsibility** | AUTOSAR |
| **Document Identification No** | 883 |

| | |
|---|---|
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Adaptive Platform |
| **Part of Standard Release** | R22-11 |

| Document Change History | | | |
|---|---|---|---|
| **Date** | **Release** | **Changed by** | **Description** |
| 2022-11-24 | R22-11 | AUTOSAR Release Management | <ul><li>Added:<br>[SWS_CRYPT_40984]<br>[SWS_CRYPT_40985]</li><li>Updated:<br>[SWS_CRYPT_00502]<br>[SWS_CRYPT_00503]<br>[SWS_CRYPT_40983]<br>[SWS_CRYPT_01820]<br>[SWS_CRYPT_01821]<br>[SWS_CRYPT_00919]<br>[SWS_CRYPT_03303]<br>[SWS_CRYPT_03305]<br>[SWS_CRYPT_04204]<br>[SWS_CRYPT_01821]<br>[SWS_CRYPT_00919]</li><li>Updated API:<br>[SWS_CRYPT_40981]<br>[SWS_CRYPT_20746]<br>[SWS_CRYPT_21115]<br>[SWS_CRYPT_21312]<br>[SWS_CRYPT_22711]<br>[SWS_CRYPT_23012]<br>[SWS_CRYPT_23013]<br>[SWS_CRYPT_23014]</li><li>Editorial changes</li></ul> |

| 2021-11-25 | R21-11 | AUTOSAR Release Management | • Reworked SWS items in chapter 7 to improve testability<br>• Improved traceability between chapters 7 and 8 |
|---|---|---|---|
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • Rewrote the document to align with AUTOSAR standard<br>• Update of Crypto API according to WG-SEC feedback |
| 2019-03-29 | R19-03 | AUTOSAR Release Management | • "Direct" prefix of Crypto API is removed, because now it is single<br>• All bugs found after R18-03 are fixed<br>• Crypto API is converted for usage of basic `ara::core` types<br>• Crypto API is converted for support of the "Exception-less" approach<br>• Detalization of Crypto API specification is extended |
| 2018-08-20 | R18-10 | AUTOSAR Release Management | • Removed crypto API introduced in release 17-10 |
| 2018-03-29 | R18-03 | AUTOSAR Release Management | • Crypto API introduced at previous release is renamed to Modeled API, chapter 7 is updated<br>• Added specification of additional Direct Crypto API (chapter 9) |
| 2017-10-27 | R17-10 | AUTOSAR Release Management | • Initial release |

**Disclaimer**

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

# Contents

# 1 Introduction and functional overview

This specification describes the functionality and the configuration for the Adaptive AUTOSAR `Functional Cluster` Cryptography (`FC Crypto`) and its API (`CryptoAPI`, which is part of the AUTOSAR Adaptive Platform Foundation.

The `FC Crypto` offers applications and other Adaptive AUTOSAR `Functional Clusters` a standardized interface, which provides operations for cryptographic and related calculations. These operations include cryptographic operations, key management, and certificate handling. `FC Crypto` manages the actual implementations of all operations, the configuration, and the brokering of operations from applications to implementations. The standardized interface is exposed by the `CryptoAPI`.

The `FC Crypto` and its `CryptoAPI` supports both public-key and symmetric-key cryptography. It allows applications to use mechanisms such as authentication, encryption, and decryption for automotive services.

# 2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the FC Crypto module that are not included in the [1, AUTOSAR glossary].

| Abbreviation / Acronym: | Description: |
| --- | --- |
| ACL | Access Control List |
| AE | Authenticated Encryption |
| AEAD | Authenticated Encryption with Associated Data – Encryption scheme which simultaneously provides confidentiality and authenticity of data as well as additional authenticated but not encrypted data. |
| AES | Advanced Encryption Standard – A block cipher for the symmetric encryption of electronic data. |
| API | Abstract Programming Interface |
| ARA | Autosar Runtime Environment for Adaptive Applications |
| ASN.1 | Abstract Syntax Notation One, as defined in the ASN.1 standards |
| BER | Basic Encoding Rules |
| BLOB | Binary Large Object – A Binary Large OBject (BLOB) is a collection of binary data stored as a single entity. |
| CA | Certificate Authority or Certification Authority is an entity that issues digital certificates. |
| CBC | Cipher Block Chaining Mode – A mode of operation for symmetric ciphers (e.g. `AES`) that supports encryption. |
| CBC-MAC | Cipher Block Chaining Message Authentication Mode – A mode of operation for symmetric ciphers (e.g. `AES`) that supports authentication. |
| CCM | Counter Mode with `CBC-MAC` – An `AEAD` operation mode (encryption and authentication) for `AES`. |
| CMAC | Cipher-based Message Authentication Code – A mode of operation for symmetric ciphers (e.g. `AES`) that supports authentication and is similar but advanced to `CBC-MAC`. |
| CMP | `X.509` Certificate Management Provider. |
| CO | Cryptographic Object |
| COUID | Cryptographic Object Unique Identifier |
| CRL | Certificate Revocation Lists is a list of digital certificates that have been revoked before their expiration date was reached. This list contains all the serial numbers of the revoked certificates and the revoked data. |
| CSR | Certificate Signing Request |
| CTL | Certificate Trust List is a list of digital certificates that are explicitly trusted in this environment. This list contains all the serial numbers of the explicitly trusted certificates. |
| DER | Distinguished Encoding Rules as defined in [2] |
| DH | Diffie-Hellman (key exchange method) |
| ECC | Elliptic Curve Cryptography – Public-key cryptography based on the structure of elliptic curves. |
| ECDH | Elliptic Curve Diffie-Hellman – An `ECC` based `DH` key exchange with perfect forward secrecy. |
| ECDSA | Elliptic Curve Digital Signature Algorithm – An `ECC` based signature scheme. |
| ECIES | Elliptic Curve Integrated Encryption Scheme – An `ECC` based encryption scheme. |
| ECU | Electronic Control Unit |

| Abbreviation / Acronym: | Description: |
|---|---|
| FC Crypto | Functional cluster Cryptography. This is the AUTOSAR cluster, which provides all important functionality related to cryptograhic, key management, and certificate handling needs. |
| gamma | linear recurrent sequence |
| GCM | Galois Counter Mode – An `AEAD` operation mode (encryption and authentication) for `AES`. |
| GMAC | Galois MAC – A mode of operation for symmetric ciphers (e.g. `AES`) that supports authentication. |
| HSM | Hardware Security Module – Hardware security module, used to store cryptographic credentials and secure run-time environment |
| HMAC | Hashed Message Authentication Code |
| IAM | Identity and Access Management |
| IETF | Internet Engineering Task Force |
| IKE | Internet Key Exchange |
| IPC | Inter-Process Communication |
| IPsec | Internet Protocol Security (IPsec) is a secure network protocol suite that authenticates and encrypts the packets of data to provide secure encrypted communication between two computers over an Internet Protocol network. |
| IV | Initialization Vector |
| KDF | Key Derivation Function – A function to derive one or more keys from a secret value. |
| KEK | Key encryption key – A key that is used to encrypt another key for transportation or storage in an unsecure environment |
| KEM | Key Encapsulation Mechanism |
| KSP | Key Storage Provider |
| MAC | Message Authentication Code |
| MGF | Mask Generation Function – A cryptographic function similar to a hash function. It takes a variable length input and an output length l to generate an output of length l. If the input is unknown, the output appears random. |
| OCSP | Online Certificate Status Protocol – Internet protocol used to obtain revocation status of `X.509` certificates. |
| PDP | Policy Decision Point |
| PEP | Policy Enforcement Point |
| PEM | Privacy-Enhanced Mail |
| PKI | Public Key Infrastructure – A system that issues, distributes, and checks digital certificates. |
| PKCS | Public Key Cryptography Standard. |
| RA | Registration Authority |
| RNG | Random Number Generator |
| RSA | Rivest, Shamir, Adleman – RSA is an algorithm for public-key cryptography; It is named after its inventors Ronald L. Rivest, Adi Shamir and Leonard Adleman. |
| SecOC | Secure Onboard Communication |
| SHA-1 | Secure Hash Algorithm (version 1) – Hash functions family. |
| SHA-2 | Secure Hash Algorithm (version 2) – Hash functions family with different hash value length. |
| SHA-3 | Secure Hash Algorithm (version 3) – New hash function generation, faster and more secure as `SHA-2`. |
| SHE | Secure Hardware Extension |
| TLS | Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network. |

| Abbreviation / Acronym: | Description: |
|---|---|
| TPM | The Trusted Platform Module is defined in [3] and is a secure cryptoprocessor. |
| UCM | Update and Configuration Management |
| UID | Unique Identifier |
| X.509 | Standard for certificates |

| Terms: | Description: |
|---|---|
| Adaptive Application | An adaptive application is a part of application SW in the architecture of Adaptive AUTOSAR. An adaptive application runs on top of ARA and accesses AUTOSAR functional clusters through ARA. |
| Adaptive Platform Services | Adaptive Platform Services are located below the ARA. They provide platform standard services of Adaptive AUTOSAR. |
| Asymmetric Key | An asymmetric key describes a pair of two keys (public and private key). A cipher text created by one key cannot be decrypted with this key. Encryption is only possible with the other key of this pair. |
| Block Cipher | A symmetric encryption that encrypts plaintext blocks of fixed length. |
| certificate serial number | An integer value, unique within the issuing authority, which is unambiguously associated with a certificate issued by that authority. |
| Certificate Slot | Secure storage of certificate material. Certificate slots define the access to the stored certificate material and may grant the access only to authorized application or functional cluster. |
| certification path | An ordered list of one or more public-key certificates, starting with a public-key certificate signed by the trust anchor, and ending with the public key certificate to be validated. All intermediate public-key certificates, if any, are CA-certificates in which the subject of the preceding certificate is the issuer of the following certificate. |
| Ciphertext | A ciphertext is an encrypted text, which is the result of encryption performed on `plaintext`. |
| CryptoAPI | The set of all interfaces that are provided by FC Crypto to consumers. |
| Crypto Provider | A structural element that organizes cryptographic primitives. |
| Cryptographic primitives | Well-established, low-level cryptographic algorithms that are frequently used to build cryptographic protocols for computer security systems. |
| Distinguished name | is originally defined in X.501 [4] as a representation of a directory name, defined as a construct that identifies a particular object from among a set of all objects. |
| Functional Cluster | The SW functionality of ARA is divided into functional clusters. Functional clusters provide APIs and can communicate with each other. |
| Identity and Access Management | A functional cluster of adaptive AUTOSAR. |
| Instance Specifier | Crypto provider can have more than one instance. To distinguish between instances the spcific instance is addressed with an instance specifier. An instance specifier identifies one instance of a crypto provider. |
| Key Material | public keys, private keys, seeds. |
| Key Slot | Secure storage of key material. Key slots define the access to the stored key material and grant the access only to authorized application or functional cluster. |

| Terms: | Description: |
|---|---|
| Key Storage Provider | A structural element that organizes and manages cryptographic keys. |
| Message Authentication Code | A cryptographic function similar to a hash function. It takes a message of variable length and a secret key as input to generate a hash value, the MAC value. The MAC value is attached to the message to be sent. The receiver of the message can recalculate the MAC value to check if the message is authentic. |
| Nonce | A nonce is a random or semi-random number that is generated for cryptographic topics. A nonce can be used as an input to a hash algorithm so that the hash algorithm computes a hash value out of two inputs: plaintext and nonce. Usage of nonces enhances security against brute force attacks. |
| Plaintext | A plaintext is ordinary readable text before being encrypted into ciphertext or after being decrypted. |
| Policy Decision Point | A PDP defines which item (process, application, function) can decide if a requested access to resources may be granted or not. |
| Policy Enforcement Point | A PEP is the point a policy decision is used to grant or deny the access. |
| Random Number Generator | A program that generates random numbers or pseudo random numbers in a given range. |
| Salt | A salt is a random or semi-random number which is created for passwords. When a password is edited for a user/account also a salt is created for this user/account. A hash algorithm creates a hash value of password and salt. Salts increase the security against brute force password guessing attacks. |
| SecretSeed | A secret value that is used as an initial value to start encryption/decryption. |
| Stream Cipher | A symmetric encryption that calculates cipher text out of streaming plaintext and the status result of the encryption of previous streamed plaintext. For the first part of encryption a start value is needed as status result. |
| Symmetric Key | In a symmetric encryption the same key (symmetric key) is used to encrypt plaintext into cipher text and to decode cipher text into plaintext. A symmetric key is also called secret key because it must be kept secret. |
| X.509 Provider | Domain SW for X.509 certificates parsing, verification, storage and search. |

# 3 Related documentation

## 3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

[2] X.690 : Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
https://www.itu.int/rec/T-REC-X.690

[3] ISO/IEC 11889-1:2015 Information technology - Trusted platform module library - Part 1: Architecture
http://www.iso.org

[4] X.501 : Information technology - Open Systems Interconnection - The Directory: Models
https://www.itu.int/rec/T-REC-X.501

[5] Specification of Adaptive Platform Core
AUTOSAR_SWS_AdaptivePlatformCore

[6] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[7] Requirements on Cryptography
AUTOSAR_RS_Cryptography

[8] BSI: Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators (AIS)
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_20_Functionality_Classes_Evaluation_Methodology_DRNG_e.pdf?__blob=publicationFile[5]

[9] Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf

[10] Public Key Cryptography for the Financial Services Industry Key Agreement and Key Stransport Using Elliptic Curve Cryptography
https://webstore.ansi.org/preview-pages/ASCX9/preview_ANSI+X9.63-2011+(R2017).pdf

[11] Recommendation for Key Derivation Using Pseudorandom Functions (Revised)
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf

[12] Elliptic Curve Cryptography
https://www.secg.org/sec1-v2.pdf

[13] ISO IEC 9797-3:2011 Amd 1:2020(en) Information technology - Security tech-

niques - Message Authentication Codes (MAC)
http://www.iso.org

[14] HMAC: Keyed-Hashing for Message Authentication
https://tools.ietf.org/html/rfc2104

[15] Updated Security Considerations the MD5 Message-Digest and the HMAC-MD5 Algorithms
https://tools.ietf.org/html/rfc6151

[16] Using Advanced Encryption Standard Counter Mode (AES-CTR) with the Internet Key Exchange version 02 (IKEv2) Protocol
https://rfc-editor.org/rfc/rfc5930.txt

[17] ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)
https://rfc-editor.org/rfc/rfc7905.txt

[18] TRIVIUM Specifications
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.9030

[19] PKCS #5: Password-Based Cryptography Specification Version 2.0
https://rfc-editor.org/rfc/rfc2898.txt

[20] PKCS #5: Password-Based Cryptography Specification Version 2.1
https://rfc-editor.org/rfc/rfc8018.txt

[21] PKCS #7: Cryptographic Message Syntax Version 1.5
https://rfc-editor.org/rfc/rfc2315.txt

[22] Financial institution encryption of wholesale financial messages: X9.23

[23] Advanced Encryption Standard (AES) Key Wrap Algorithm
https://tools.ietf.org/html/rfc3394

[24] Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm
https://tools.ietf.org/html/rfc5649

[25] ISO/IEC 9796-2:2010 Information technology - Security techniques - Digital signature schemes giving message recovery - Part 2: Integer factorization based mechanisms
http://www.iso.org

[26] Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)
https://rfc-editor.org/rfc/rfc3278.txt

[27] Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)
https://rfc-editor.org/rfc/rfc5753.txt

[28] IEEE P1363: A Standard for RSA, Diffie-Hellman, and Elliptic-Curve Cryptography (Abstract)

[29] New directions in cryptography
https://ieeexplore.ieee.org/document/1055638

[30] Guide for Internet Standards Writers
https://tools.ietf.org/html/rfc2360

[31] X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP
https://rfc-editor.org/rfc/rfc6960.txt

[32] Standard X.509
https://www.itu.int/rec/T-REC-X.509/en

[33] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List
(CRL) Profile
https://rfc-editor.org/rfc/rfc5280.txt

[34] PKCS #10: Certification Request Syntax Specification Version 1.7
https://tools.ietf.org/html/rfc2986

[35] The application/pkcs10 Media Type
https://tools.ietf.org/html/rfc5967

[36] Internet X.509 Certificate Request Message Format
https://tools.ietf.org/html/rfc2511

[37] Internet X.509 Public Key Infrastructure Certificate Request Message Format
(CRMF)
https://tools.ietf.org/html/rfc4211

[38] S/MIME Version 2 Message Specification
https://tools.ietf.org/html/rfc2311

[39] Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax
Specification Version 1.2
https://rfc-editor.org/rfc/rfc5208.txt

[40] PKCS #12: Personal Information Exchange Syntax v1.1
https://tools.ietf.org/html/rfc7292

[41] X.680 : Information technology - Abstract Syntax Notation One (ASN.1): Specifi-
cation of basic notation
https://www.itu.int/rec/T-REC-X.680

[42] X.682 : Information technology - Abstract Syntax Notation One (ASN.1): Con-
straint specification
https://www.itu.int/rec/T-REC-X.682

[43] X.683 : Information technology - Abstract Syntax Notation One (ASN.1): Param-
eterization of ASN.1 specifications
https://www.itu.int/rec/T-REC-X.683

[44] Keying and Authentication for Routing Protocols (KARP) Design Guidelines
https://tools.ietf.org/html/rfc6518

[45]  Internationalized Email Addresses in X.509 Certificates
https://tools.ietf.org/html/rfc8398

[46]  Internationalization Updates to RFC 5280
https://tools.ietf.org/html/rfc8399

[47]  Transport Layer Security (TLS) Extensions: Extension Definitions
https://tools.ietf.org/html/rfc6066

[48]  The Transport Layer Security (TLS) Multiple Certificate Status Request Extension
https://tools.ietf.org/html/rfc6961

[49]  The Transport Layer Security (TLS) Protocol Version 1.3
https://tools.ietf.org/html/rfc8446

## 3.2   Further applicable specification

AUTOSAR provides a core specification [5, SWS AdaptivePlatformCore] which is also applicable for FC Crypto.  The chapter "General requirements for all FunctionalClusters" of this specification shall be considered as an additional and required specification for implementation of FC Crypto.

# 4 Constraints and assumptions

## 4.1 Constraints

For the design of the `FC Crypto` and the `CryptoAPI` the following constraints were applied:

- Support the independence of application software components from a specific platform implementation.

- Make the API as lean as possible, no specific use cases are supported, which could also be layered on top of the API.

- Offer a "comfort layer" to enable the use of C++11/14 features.

- Support the integration into safety relevant systems.

- Support the integration into cyber security relevant systems.

## 4.2 Assumptions

The `Adaptive Application` and `Functional Cluster` should not have direct access to keys within its own process. The `FC Crypto` and its building blocks mediates for `Adaptive Application` and `Functional Cluster` access and usage of secret `Key Material`. Therefore, the `FC Crypto` verifies whether an application or functional cluster is allowed to access a specific cryptographic object, which is stored in the infrastructure of the `FC Crypto`. This access control mechanism is realized in combination with `IAM`, where the `FC Crypto` acts as a policy enforcment point.

Beside the support of applications and `Functional Clusters`, the `FC Crypto` provides mechanism to ensure secure communication. The `FC Crypto` helps `Adaptive Application` and `Functional Cluster` to establish secure channels. The `FC Crypto` also allows to store data persistent in an encrypted manner.

## 4.3 Known limitations

The following functional domains and descriptions are still missing in the current version of Crypto API specification:

- **Asynchronous interfaces**
  Currently there is only a synchronous API specification and asynchronous behavior (if required) should be implemented on the consumer application level. It can be done via utilization of dedicated execution threads for long-time operations.

- **Full X.509 certificate support incl. OCSP and OCSP stabling**
  `CryptoAPI` doesn't provide complete specification of the X.509 certificates man-

agement on the client (ECU) side yet. Current version of Crypto API specifies only minimal subset of interfaces responsible for basic X.509 functionality and related on utilization of cryptographic algorithms. Current API supports extraction and parsing of only basic attributes of X.509 certificates and certification requests. An extension of the API specification by additional interfaces dedicated for complete support of X.509 extensions is planned for the next release of this specification. **Note:** Generally current specification of the X.509 Provider API is preliminary and subject for extensions and changes.

- **Formats of certificate objects**
  Current version of `CryptoAPI` has minimal support of well-known cryptographic formats encoding/decoding: support of only DER and PEM encoding for X.509 certificates and certificate signing requests is required from any implementation of `CryptoAPI`. For other cryptographic objects an implementation can support only "raw" formats. Following extension of the `CryptoAPI` by unified interfaces for encoding/decoding of complex objects to standard formats is planned for the next release of this specification.

## 4.4 Applicability to car domains

No restrictions to applicability.

# 5 Dependencies to other functional clusters

There is a dependency to IAM that concerns PEP and PDP. For details see 7.2.1.

## 5.1 Protocol layer dependencies

There are currently no dependencies to protocol layers.

# 6 Requirements Tracing

The following tables reference the requirements specified in [6] and [7] and links to the fulfillment of these. Please note that if column "Satisfied by" is empty for a specific requirement this means that this requirement is not fulfilled by this document.

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_AP_00130]** | AUTOSAR Adaptive Platform shall represent a rich and modern programming environment. | [SWS_CRYPT_19900] |
| **[RS_AP_00144]** | Availability of a named constructor. | [SWS_CRYPT_20745]<br>[SWS_CRYPT_20746]<br>[SWS_CRYPT_20747]<br>[SWS_CRYPT_20748]<br>[SWS_CRYPT_20750]<br>[SWS_CRYPT_20751]<br>[SWS_CRYPT_20752]<br>[SWS_CRYPT_20753]<br>[SWS_CRYPT_20754]<br>[SWS_CRYPT_20755]<br>[SWS_CRYPT_20756]<br>[SWS_CRYPT_20757]<br>[SWS_CRYPT_20758]<br>[SWS_CRYPT_20760]<br>[SWS_CRYPT_20761] |
| **[RS_CRYPTO_-02001]** | The Crypto Stack shall conceal symmetric keys from the users | [SWS_CRYPT_00007]<br>[SWS_CRYPT_20733]<br>[SWS_CRYPT_20762]<br>[SWS_CRYPT_20763]<br>[SWS_CRYPT_20764]<br>[SWS_CRYPT_20765]<br>[SWS_CRYPT_20810]<br>[SWS_CRYPT_21010]<br>[SWS_CRYPT_21313]<br>[SWS_CRYPT_21413]<br>[SWS_CRYPT_21525]<br>[SWS_CRYPT_21815]<br>[SWS_CRYPT_22118]<br>[SWS_CRYPT_22211]<br>[SWS_CRYPT_22913]<br>[SWS_CRYPT_23211]<br>[SWS_CRYPT_23515]<br>[SWS_CRYPT_23623]<br>[SWS_CRYPT_23710]<br>[SWS_CRYPT_23800]<br>[SWS_CRYPT_23911]<br>[SWS_CRYPT_24018]<br>[SWS_CRYPT_24115] |

| Requirement | Description | Satisfied by |
|---|---|---|
| [RS_CRYPTO_-02002] | The Crypto Stack shall conceal asymmetric private keys from the users | [SWS_CRYPT_00007]<br>[SWS_CRYPT_10305]<br>[SWS_CRYPT_20733]<br>[SWS_CRYPT_20762]<br>[SWS_CRYPT_20763]<br>[SWS_CRYPT_20764]<br>[SWS_CRYPT_20765]<br>[SWS_CRYPT_22500] |
| [RS_CRYPTO_-02003] | The Crypto Stack shall support management of non-persistent session/ephemeral keys during their lifetime | [SWS_CRYPT_20512]<br>[SWS_CRYPT_20721]<br>[SWS_CRYPT_20722]<br>[SWS_CRYPT_20810]<br>[SWS_CRYPT_21010]<br>[SWS_CRYPT_21313]<br>[SWS_CRYPT_21413]<br>[SWS_CRYPT_21525]<br>[SWS_CRYPT_21815]<br>[SWS_CRYPT_22118]<br>[SWS_CRYPT_22211]<br>[SWS_CRYPT_22913]<br>[SWS_CRYPT_23211]<br>[SWS_CRYPT_23515]<br>[SWS_CRYPT_23623]<br>[SWS_CRYPT_23710]<br>[SWS_CRYPT_23911]<br>[SWS_CRYPT_24018]<br>[SWS_CRYPT_24115] |
| [RS_CRYPTO_-02004] | The Crypto Stack shall support secure storage of cryptographic artifacts | [SWS_CRYPT_00102]<br>[SWS_CRYPT_00103]<br>[SWS_CRYPT_04202]<br>[SWS_CRYPT_04203]<br>[SWS_CRYPT_04204]<br>[SWS_CRYPT_04205]<br>[SWS_CRYPT_04207]<br>[SWS_CRYPT_04208]<br>[SWS_CRYPT_04209]<br>[SWS_CRYPT_10000]<br>[SWS_CRYPT_10016]<br>[SWS_CRYPT_10018]<br>[SWS_CRYPT_10019]<br>[SWS_CRYPT_10031]<br>[SWS_CRYPT_10033]<br>[SWS_CRYPT_10701]<br>[SWS_CRYPT_10710]<br>[SWS_CRYPT_10750]<br>[SWS_CRYPT_10751]<br>[SWS_CRYPT_10752]<br>[SWS_CRYPT_10753]<br>[SWS_CRYPT_10800]<br>[SWS_CRYPT_10810]<br>[SWS_CRYPT_10811] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_10818] |
| | | [SWS_CRYPT_10821] |
| | | [SWS_CRYPT_10822] |
| | | [SWS_CRYPT_10823] |
| | | [SWS_CRYPT_10850] |
| | | [SWS_CRYPT_10851] |
| | | [SWS_CRYPT_10852] |
| | | [SWS_CRYPT_10853] |
| | | [SWS_CRYPT_20517] |
| | | [SWS_CRYPT_30010] |
| | | [SWS_CRYPT_30011] |
| | | [SWS_CRYPT_30101] |
| | | [SWS_CRYPT_30110] |
| | | [SWS_CRYPT_30115] |
| | | [SWS_CRYPT_30123] |
| | | [SWS_CRYPT_30124] |
| | | [SWS_CRYPT_30125] |
| | | [SWS_CRYPT_30126] |
| | | [SWS_CRYPT_30200] |
| | | [SWS_CRYPT_30201] |
| | | [SWS_CRYPT_30202] |
| | | [SWS_CRYPT_30203] |
| | | [SWS_CRYPT_30204] |
| | | [SWS_CRYPT_30205] |
| | | [SWS_CRYPT_30206] |
| | | [SWS_CRYPT_30207] |
| | | [SWS_CRYPT_30209] |
| | | [SWS_CRYPT_30210] |
| | | [SWS_CRYPT_30211] |
| | | [SWS_CRYPT_30212] |
| | | [SWS_CRYPT_30213] |
| | | [SWS_CRYPT_30214] |
| | | [SWS_CRYPT_30215] |
| | | [SWS_CRYPT_30216] |
| | | [SWS_CRYPT_30217] |
| | | [SWS_CRYPT_30218] |
| | | [SWS_CRYPT_30219] |
| | | [SWS_CRYPT_30220] |
| | | [SWS_CRYPT_30221] |
| | | [SWS_CRYPT_30222] |
| | | [SWS_CRYPT_30223] |
| | | [SWS_CRYPT_30224] |
| | | [SWS_CRYPT_30225] |
| | | [SWS_CRYPT_30226] |
| | | [SWS_CRYPT_30227] |
| | | [SWS_CRYPT_30404] |
| | | [SWS_CRYPT_30406] |
| | | [SWS_CRYPT_30408] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_30409] |
| | | [SWS_CRYPT_40947] |
| | | [SWS_CRYPT_40948] |
| | | [SWS_CRYPT_40949] |
| | | [SWS_CRYPT_40950] |
| | | [SWS_CRYPT_40951] |
| | | [SWS_CRYPT_40952] |
| | | [SWS_CRYPT_40953] |
| | | [SWS_CRYPT_40954] |
| | | [SWS_CRYPT_40955] |
| | | [SWS_CRYPT_40956] |
| | | [SWS_CRYPT_40957] |
| | | [SWS_CRYPT_40959] |
| [RS_CRYPTO_-02005] | The Crypto Stack shall support unique identification of cryptographic objects | [SWS_CRYPT_10100] |
| | | [SWS_CRYPT_10150] |
| | | [SWS_CRYPT_10151] |
| | | [SWS_CRYPT_10152] |
| | | [SWS_CRYPT_10153] |
| | | [SWS_CRYPT_10154] |
| | | [SWS_CRYPT_10155] |
| | | [SWS_CRYPT_10306] |
| | | [SWS_CRYPT_10400] |
| | | [SWS_CRYPT_10411] |
| | | [SWS_CRYPT_10412] |
| | | [SWS_CRYPT_10413] |
| | | [SWS_CRYPT_10808] |
| | | [SWS_CRYPT_20500] |
| | | [SWS_CRYPT_20501] |
| | | [SWS_CRYPT_20502] |
| | | [SWS_CRYPT_20503] |
| | | [SWS_CRYPT_20504] |
| | | [SWS_CRYPT_20505] |
| | | [SWS_CRYPT_20506] |
| | | [SWS_CRYPT_20507] |
| | | [SWS_CRYPT_20513] |
| | | [SWS_CRYPT_20514] |
| | | [SWS_CRYPT_20515] |
| | | [SWS_CRYPT_20518] |
| | | [SWS_CRYPT_20600] |
| | | [SWS_CRYPT_20641] |
| | | [SWS_CRYPT_20643] |
| | | [SWS_CRYPT_20644] |
| | | [SWS_CRYPT_20703] |
| | | [SWS_CRYPT_20724] |
| | | [SWS_CRYPT_20725] |
| | | [SWS_CRYPT_20726] |
| | | [SWS_CRYPT_20727] |
| | | [SWS_CRYPT_20733] |
| | | [SWS_CRYPT_20760] |
| | | [SWS_CRYPT_20761] |
| | | [SWS_CRYPT_30500] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02006]** | The Crypto Stack shall support a version control mechanism and distinguish "versions" and "origin sources" of cryptographic objects | [SWS_CRYPT_04213]<br>[SWS_CRYPT_10100]<br>[SWS_CRYPT_10101]<br>[SWS_CRYPT_10102]<br>[SWS_CRYPT_10111]<br>[SWS_CRYPT_10112]<br>[SWS_CRYPT_10113]<br>[SWS_CRYPT_10114]<br>[SWS_CRYPT_10115]<br>[SWS_CRYPT_20102]<br>[SWS_CRYPT_20703]<br>[SWS_CRYPT_20724]<br>[SWS_CRYPT_20725]<br>[SWS_CRYPT_20726]<br>[SWS_CRYPT_20727]<br>[SWS_CRYPT_20733]<br>[SWS_CRYPT_20760]<br>[SWS_CRYPT_20761]<br>[SWS_CRYPT_20802]<br>[SWS_CRYPT_21002]<br>[SWS_CRYPT_21102]<br>[SWS_CRYPT_21302]<br>[SWS_CRYPT_21402]<br>[SWS_CRYPT_21517]<br>[SWS_CRYPT_21802]<br>[SWS_CRYPT_22102]<br>[SWS_CRYPT_22210]<br>[SWS_CRYPT_22902]<br>[SWS_CRYPT_23210]<br>[SWS_CRYPT_23510]<br>[SWS_CRYPT_23602]<br>[SWS_CRYPT_23702]<br>[SWS_CRYPT_24002]<br>[SWS_CRYPT_24102]<br>[SWS_CRYPT_40958] |
| **[RS_CRYPTO_-02007]** | The Crypto Stack shall provide means for secure handling of "secret seeds" | [SWS_CRYPT_00102]<br>[SWS_CRYPT_10401]<br>[SWS_CRYPT_20723]<br>[SWS_CRYPT_21311]<br>[SWS_CRYPT_21411]<br>[SWS_CRYPT_21516]<br>[SWS_CRYPT_21810]<br>[SWS_CRYPT_23000]<br>[SWS_CRYPT_23001]<br>[SWS_CRYPT_23002]<br>[SWS_CRYPT_23003]<br>[SWS_CRYPT_23011]<br>[SWS_CRYPT_23012]<br>[SWS_CRYPT_23013]<br>[SWS_CRYPT_23014]<br>[SWS_CRYPT_23015]<br>[SWS_CRYPT_23016]<br>[SWS_CRYPT_24015] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02008]** | The Crypto Stack shall support restrictions of the allowed usage scope for keys and "secret seeds" | [SWS_CRYPT_10004]<br>[SWS_CRYPT_10819]<br>[SWS_CRYPT_20400]<br>[SWS_CRYPT_20401]<br>[SWS_CRYPT_20402]<br>[SWS_CRYPT_20411]<br>[SWS_CRYPT_21521]<br>[SWS_CRYPT_24800]<br>[SWS_CRYPT_24801]<br>[SWS_CRYPT_24811]<br>[SWS_CRYPT_29046] |
| **[RS_CRYPTO_-02009]** | The Crypto stack shall support separation of applications" access rights for each cryptographic object slot | [SWS_CRYPT_10003]<br>[SWS_CRYPT_10004]<br>[SWS_CRYPT_30208]<br>[SWS_CRYPT_30300]<br>[SWS_CRYPT_30405] |
| **[RS_CRYPTO_-02101]** | The Crypto Stack shall provide interfaces to generate cryptographic keys for all supported primitives | [SWS_CRYPT_00601]<br>[SWS_CRYPT_00603]<br>[SWS_CRYPT_00608]<br>[SWS_CRYPT_00609]<br>[SWS_CRYPT_00610]<br>[SWS_CRYPT_00611]<br>[SWS_CRYPT_00622]<br>[SWS_CRYPT_03300]<br>[SWS_CRYPT_03311]<br>[SWS_CRYPT_10300]<br>[SWS_CRYPT_10301]<br>[SWS_CRYPT_10303]<br>[SWS_CRYPT_10304]<br>[SWS_CRYPT_20721]<br>[SWS_CRYPT_20722]<br>[SWS_CRYPT_40944]<br>[SWS_CRYPT_40945]<br>[SWS_CRYPT_40946]<br>[SWS_CRYPT_40962]<br>[SWS_CRYPT_40969] |
| **[RS_CRYPTO_-02102]** | The Crypto Stack shall prevent keys from being used in incompatible or insecure ways | [SWS_CRYPT_00102]<br>[SWS_CRYPT_03312]<br>[SWS_CRYPT_10014]<br>[SWS_CRYPT_20721]<br>[SWS_CRYPT_20722]<br>[SWS_CRYPT_21412]<br>[SWS_CRYPT_21512]<br>[SWS_CRYPT_21513]<br>[SWS_CRYPT_21515]<br>[SWS_CRYPT_21523]<br>[SWS_CRYPT_21813] |

| Requirement | Description | Satisfied by |
|---|---|---|
| [RS_CRYPTO_-02103] | The Crypto Stack shall support primitives to derive cryptographic key material from a base key material | [SWS_CRYPT_03313]<br>[SWS_CRYPT_10402]<br>[SWS_CRYPT_20748]<br>[SWS_CRYPT_21500]<br>[SWS_CRYPT_21501]<br>[SWS_CRYPT_21519]<br>[SWS_CRYPT_21520]<br>[SWS_CRYPT_21522] |
| [RS_CRYPTO_-02104] | The Crypto Stack shall support a primitive to exchange cryptographic keys with another entity | [SWS_CRYPT_03301]<br>[SWS_CRYPT_20743]<br>[SWS_CRYPT_20752]<br>[SWS_CRYPT_20753]<br>[SWS_CRYPT_20758]<br>[SWS_CRYPT_21300]<br>[SWS_CRYPT_21301]<br>[SWS_CRYPT_21400]<br>[SWS_CRYPT_21401]<br>[SWS_CRYPT_21800]<br>[SWS_CRYPT_24000] |
| [RS_CRYPTO_-02105] | Symmetric keys and asymmetric private keys shall be imported and exported in a secure format. | [SWS_CRYPT_03302]<br>[SWS_CRYPT_03303]<br>[SWS_CRYPT_03304]<br>[SWS_CRYPT_04200]<br>[SWS_CRYPT_10403]<br>[SWS_CRYPT_10700]<br>[SWS_CRYPT_20728]<br>[SWS_CRYPT_20729]<br>[SWS_CRYPT_20730]<br>[SWS_CRYPT_20731]<br>[SWS_CRYPT_20732] |
| [RS_CRYPTO_-02106] | The Crypto Stack shall provide interfaces for secure processing of passwords | [SWS_CRYPT_10004] |
| [RS_CRYPTO_-02107] | The Crypto Stack shall support the algorithm specification in any key generation or derivation request | [SWS_CRYPT_01501]<br>[SWS_CRYPT_01506]<br>[SWS_CRYPT_01508]<br>[SWS_CRYPT_01651]<br>[SWS_CRYPT_02123]<br>[SWS_CRYPT_10014]<br>[SWS_CRYPT_13000]<br>[SWS_CRYPT_13001]<br>[SWS_CRYPT_13002]<br>[SWS_CRYPT_13003]<br>[SWS_CRYPT_20710]<br>[SWS_CRYPT_20721]<br>[SWS_CRYPT_20722]<br>[SWS_CRYPT_21512]<br>[SWS_CRYPT_21513]<br>[SWS_CRYPT_21515]<br>[SWS_CRYPT_21523]<br>[SWS_CRYPT_40964] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02108]** | The Crypto Stack shall provide interfaces for management and usage of algorithm-specific domain parameters | [SWS_CRYPT_20414]<br>[SWS_CRYPT_20721]<br>[SWS_CRYPT_20722]<br>[SWS_CRYPT_21314]<br>[SWS_CRYPT_21412]<br>[SWS_CRYPT_21414]<br>[SWS_CRYPT_21512]<br>[SWS_CRYPT_21513]<br>[SWS_CRYPT_21515]<br>[SWS_CRYPT_21523]<br>[SWS_CRYPT_21524]<br>[SWS_CRYPT_21813]<br>[SWS_CRYPT_21816]<br>[SWS_CRYPT_22120]<br>[SWS_CRYPT_22212]<br>[SWS_CRYPT_22511]<br>[SWS_CRYPT_23212]<br>[SWS_CRYPT_23516]<br>[SWS_CRYPT_23627]<br>[SWS_CRYPT_23712]<br>[SWS_CRYPT_24019]<br>[SWS_CRYPT_24116]<br>[SWS_CRYPT_24414] |
| **[RS_CRYPTO_-02109]** | The Crypto Stack shall support interfaces for a unified Machine-wide storage and retrieval of different crypto objects | [SWS_CRYPT_10017]<br>[SWS_CRYPT_10801]<br>[SWS_CRYPT_10802]<br>[SWS_CRYPT_10814]<br>[SWS_CRYPT_10815]<br>[SWS_CRYPT_10816]<br>[SWS_CRYPT_10817]<br>[SWS_CRYPT_20701]<br>[SWS_CRYPT_30099]<br>[SWS_CRYPT_30100] |
| **[RS_CRYPTO_-02110]** | The Crypto Stack shall support prototyping of application-exclusive key slot resources | [SWS_CRYPT_00101]<br>[SWS_CRYPT_10812]<br>[SWS_CRYPT_10813]<br>[SWS_CRYPT_10818]<br>[SWS_CRYPT_30300]<br>[SWS_CRYPT_30301]<br>[SWS_CRYPT_30302]<br>[SWS_CRYPT_30305]<br>[SWS_CRYPT_30306]<br>[SWS_CRYPT_30307]<br>[SWS_CRYPT_30308]<br>[SWS_CRYPT_30309]<br>[SWS_CRYPT_30310]<br>[SWS_CRYPT_30311]<br>[SWS_CRYPT_30312]<br>[SWS_CRYPT_30313]<br>[SWS_CRYPT_30350]<br>[SWS_CRYPT_30351]<br>[SWS_CRYPT_30407] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02111]** | The Crypto Stack shall provide applications a possibility to define usage restrictions of any new generated or derived key | [SWS_CRYPT_10015]<br>[SWS_CRYPT_13100]<br>[SWS_CRYPT_13101]<br>[SWS_CRYPT_13102]<br>[SWS_CRYPT_13103]<br>[SWS_CRYPT_13104]<br>[SWS_CRYPT_13105]<br>[SWS_CRYPT_13106]<br>[SWS_CRYPT_13107]<br>[SWS_CRYPT_13108]<br>[SWS_CRYPT_13109]<br>[SWS_CRYPT_13110]<br>[SWS_CRYPT_13111]<br>[SWS_CRYPT_13112]<br>[SWS_CRYPT_13113]<br>[SWS_CRYPT_13114]<br>[SWS_CRYPT_13115]<br>[SWS_CRYPT_13116]<br>[SWS_CRYPT_13117]<br>[SWS_CRYPT_13118]<br>[SWS_CRYPT_13119]<br>[SWS_CRYPT_13120]<br>[SWS_CRYPT_13121]<br>[SWS_CRYPT_13122]<br>[SWS_CRYPT_20721]<br>[SWS_CRYPT_20722]<br>[SWS_CRYPT_21512]<br>[SWS_CRYPT_21513]<br>[SWS_CRYPT_21515]<br>[SWS_CRYPT_21523]<br>[SWS_CRYPT_30500]<br>[SWS_CRYPT_30501]<br>[SWS_CRYPT_30503]<br>[SWS_CRYPT_30505]<br>[SWS_CRYPT_30506]<br>[SWS_CRYPT_30508]<br>[SWS_CRYPT_30510]<br>[SWS_CRYPT_30511]<br>[SWS_CRYPT_30550]<br>[SWS_CRYPT_30551] |

| Requirement | Description | Satisfied by |
|---|---|---|
| [RS_CRYPTO_-02112] | The Crypto Stack shall execute export/import of a key value together with its meta information | [SWS_CRYPT_04200]<br>[SWS_CRYPT_10200]<br>[SWS_CRYPT_10451]<br>[SWS_CRYPT_10452]<br>[SWS_CRYPT_10453]<br>[SWS_CRYPT_10454]<br>[SWS_CRYPT_10455]<br>[SWS_CRYPT_10456]<br>[SWS_CRYPT_10711]<br>[SWS_CRYPT_10712]<br>[SWS_CRYPT_20005]<br>[SWS_CRYPT_20728]<br>[SWS_CRYPT_20729]<br>[SWS_CRYPT_20730]<br>[SWS_CRYPT_20731]<br>[SWS_CRYPT_20732] |
| [RS_CRYPTO_-02113] | The Crypto Stack interfaces shall support control of the exportability property of a key object | [SWS_CRYPT_04200] |
| [RS_CRYPTO_-02115] | The Crypto Stack shall enforce assigning required domain parameters to a key in its generation or derivation procedure | [SWS_CRYPT_03305]<br>[SWS_CRYPT_20721]<br>[SWS_CRYPT_20722]<br>[SWS_CRYPT_21312]<br>[SWS_CRYPT_21315]<br>[SWS_CRYPT_21412]<br>[SWS_CRYPT_21515]<br>[SWS_CRYPT_21523]<br>[SWS_CRYPT_21813]<br>[SWS_CRYPT_22511]<br>[SWS_CRYPT_24016]<br>[SWS_CRYPT_24017] |
| [RS_CRYPTO_-02116] | The Crypto Stack shall support version control of key objects kept in the Key Storage | [SWS_CRYPT_30300] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02201]** | The Crypto Stack shall provide interfaces to use symmetric encryption and decryption primitives | [SWS_CRYPT_01501]<br>[SWS_CRYPT_01502]<br>[SWS_CRYPT_01503]<br>[SWS_CRYPT_01504]<br>[SWS_CRYPT_01506]<br>[SWS_CRYPT_01508]<br>[SWS_CRYPT_01651]<br>[SWS_CRYPT_01653]<br>[SWS_CRYPT_01654]<br>[SWS_CRYPT_01655]<br>[SWS_CRYPT_01656]<br>[SWS_CRYPT_01657]<br>[SWS_CRYPT_01658]<br>[SWS_CRYPT_01659]<br>[SWS_CRYPT_01660]<br>[SWS_CRYPT_01661]<br>[SWS_CRYPT_01662]<br>[SWS_CRYPT_02123]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20742]<br>[SWS_CRYPT_20744]<br>[SWS_CRYPT_23600]<br>[SWS_CRYPT_23601]<br>[SWS_CRYPT_23700]<br>[SWS_CRYPT_23701]<br>[SWS_CRYPT_23716]<br>[SWS_CRYPT_23717]<br>[SWS_CRYPT_23801]<br>[SWS_CRYPT_23802]<br>[SWS_CRYPT_24001]<br>[SWS_CRYPT_24011]<br>[SWS_CRYPT_24012]<br>[SWS_CRYPT_24013]<br>[SWS_CRYPT_24014]<br>[SWS_CRYPT_40963]<br>[SWS_CRYPT_40964] |

Document ID 883: AUTOSAR_SWS_Cryptography

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02202]** | The Crypto Stack shall provide interfaces to use asymmetric encryption and decryption primitives | [SWS_CRYPT_02700]<br>[SWS_CRYPT_02701]<br>[SWS_CRYPT_02702]<br>[SWS_CRYPT_02703]<br>[SWS_CRYPT_02704]<br>[SWS_CRYPT_02705]<br>[SWS_CRYPT_02706]<br>[SWS_CRYPT_02726]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20750]<br>[SWS_CRYPT_20751]<br>[SWS_CRYPT_20754]<br>[SWS_CRYPT_20755]<br>[SWS_CRYPT_20800]<br>[SWS_CRYPT_20801]<br>[SWS_CRYPT_20811]<br>[SWS_CRYPT_20812]<br>[SWS_CRYPT_20813]<br>[SWS_CRYPT_21000]<br>[SWS_CRYPT_21001]<br>[SWS_CRYPT_21011]<br>[SWS_CRYPT_21012]<br>[SWS_CRYPT_21013]<br>[SWS_CRYPT_22200]<br>[SWS_CRYPT_22700]<br>[SWS_CRYPT_22701]<br>[SWS_CRYPT_22702]<br>[SWS_CRYPT_22711]<br>[SWS_CRYPT_22712]<br>[SWS_CRYPT_22713]<br>[SWS_CRYPT_23200]<br>[SWS_CRYPT_23201]<br>[SWS_CRYPT_23215]<br>[SWS_CRYPT_23216]<br>[SWS_CRYPT_40966] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02203]** | The Crypto Stack shall provide interfaces to use message authentication code primitives | [SWS_CRYPT_01200]<br>[SWS_CRYPT_01201]<br>[SWS_CRYPT_01202]<br>[SWS_CRYPT_01203]<br>[SWS_CRYPT_01204]<br>[SWS_CRYPT_01207]<br>[SWS_CRYPT_01208]<br>[SWS_CRYPT_01209]<br>[SWS_CRYPT_01210]<br>[SWS_CRYPT_01211]<br>[SWS_CRYPT_01213]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20319]<br>[SWS_CRYPT_20746]<br>[SWS_CRYPT_22100]<br>[SWS_CRYPT_22101]<br>[SWS_CRYPT_22115]<br>[SWS_CRYPT_22116]<br>[SWS_CRYPT_22117]<br>[SWS_CRYPT_22119]<br>[SWS_CRYPT_23300]<br>[SWS_CRYPT_23301]<br>[SWS_CRYPT_23302]<br>[SWS_CRYPT_23311] |
| **[RS_CRYPTO_-02204]** | The Crypto Stack shall provide interfaces to use digital signature primitives | [SWS_CRYPT_00902]<br>[SWS_CRYPT_02400]<br>[SWS_CRYPT_02408]<br>[SWS_CRYPT_02409]<br>[SWS_CRYPT_02410]<br>[SWS_CRYPT_02411]<br>[SWS_CRYPT_02412]<br>[SWS_CRYPT_02413]<br>[SWS_CRYPT_02414]<br>[SWS_CRYPT_02415]<br>[SWS_CRYPT_02416]<br>[SWS_CRYPT_02417]<br>[SWS_CRYPT_02418]<br>[SWS_CRYPT_02419]<br>[SWS_CRYPT_02420]<br>[SWS_CRYPT_02421]<br>[SWS_CRYPT_02422]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20003]<br>[SWS_CRYPT_20319]<br>[SWS_CRYPT_20754]<br>[SWS_CRYPT_20755]<br>[SWS_CRYPT_20756]<br>[SWS_CRYPT_20757] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_22119] |
| | | [SWS_CRYPT_22200] |
| | | [SWS_CRYPT_22201] |
| | | [SWS_CRYPT_22215] |
| | | [SWS_CRYPT_22216] |
| | | [SWS_CRYPT_23200] |
| | | [SWS_CRYPT_23201] |
| | | [SWS_CRYPT_23300] |
| | | [SWS_CRYPT_23301] |
| | | [SWS_CRYPT_23302] |
| | | [SWS_CRYPT_23311] |
| | | [SWS_CRYPT_23500] |
| | | [SWS_CRYPT_23501] |
| | | [SWS_CRYPT_23511] |
| | | [SWS_CRYPT_23512] |
| | | [SWS_CRYPT_23513] |
| | | [SWS_CRYPT_23514] |
| | | [SWS_CRYPT_24100] |
| | | [SWS_CRYPT_24101] |
| | | [SWS_CRYPT_24111] |
| | | [SWS_CRYPT_24112] |
| | | [SWS_CRYPT_24113] |
| | | [SWS_CRYPT_24114] |
| | | [SWS_CRYPT_40961] |
| [RS_CRYPTO_-02205] | The Crypto Stack shall provide interfaces to use hashing primitives | [SWS_CRYPT_00901] |
| | | [SWS_CRYPT_00903] |
| | | [SWS_CRYPT_00905] |
| | | [SWS_CRYPT_00906] |
| | | [SWS_CRYPT_00907] |
| | | [SWS_CRYPT_00908] |
| | | [SWS_CRYPT_00909] |
| | | [SWS_CRYPT_00910] |
| | | [SWS_CRYPT_00919] |
| | | [SWS_CRYPT_10042] |
| | | [SWS_CRYPT_20747] |
| | | [SWS_CRYPT_21100] |
| | | [SWS_CRYPT_21101] |
| | | [SWS_CRYPT_21115] |
| | | [SWS_CRYPT_21116] |
| | | [SWS_CRYPT_21117] |
| | | [SWS_CRYPT_23300] |
| | | [SWS_CRYPT_23301] |
| | | [SWS_CRYPT_23302] |
| | | [SWS_CRYPT_23311] |
| | | [SWS_CRYPT_40960] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02206]** | The Crypto Stack shall provide interfaces to configure and use random number generation | [SWS_CRYPT_00500]<br>[SWS_CRYPT_00501]<br>[SWS_CRYPT_00502]<br>[SWS_CRYPT_00503]<br>[SWS_CRYPT_00504]<br>[SWS_CRYPT_00505]<br>[SWS_CRYPT_00506]<br>[SWS_CRYPT_00507]<br>[SWS_CRYPT_00508]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20741]<br>[SWS_CRYPT_22900]<br>[SWS_CRYPT_22901]<br>[SWS_CRYPT_22911]<br>[SWS_CRYPT_22912]<br>[SWS_CRYPT_22914]<br>[SWS_CRYPT_22915]<br>[SWS_CRYPT_30098]<br>[SWS_CRYPT_40983] |
| **[RS_CRYPTO_-02207]** | The Crypto Stack shall provide interfaces to use authenticated symmetric encryption and decryption primitives | [SWS_CRYPT_01800]<br>[SWS_CRYPT_01801]<br>[SWS_CRYPT_01802]<br>[SWS_CRYPT_01803]<br>[SWS_CRYPT_01804]<br>[SWS_CRYPT_01805]<br>[SWS_CRYPT_01806]<br>[SWS_CRYPT_01807]<br>[SWS_CRYPT_01808]<br>[SWS_CRYPT_01811]<br>[SWS_CRYPT_01820]<br>[SWS_CRYPT_01821]<br>[SWS_CRYPT_01822]<br>[SWS_CRYPT_01823]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20100]<br>[SWS_CRYPT_20101]<br>[SWS_CRYPT_20316]<br>[SWS_CRYPT_20745] |
| **[RS_CRYPTO_-02208]** | The Crypto Stack shall provide interfaces to use symmetric key wrapping primitives | [SWS_CRYPT_02104]<br>[SWS_CRYPT_02105]<br>[SWS_CRYPT_02106]<br>[SWS_CRYPT_02107]<br>[SWS_CRYPT_02108]<br>[SWS_CRYPT_02109]<br>[SWS_CRYPT_02121]<br>[SWS_CRYPT_02122]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20743]<br>[SWS_CRYPT_24000]<br>[SWS_CRYPT_40965] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02209]** | The Crypto Stack shall provide interfaces to use asymmetric key encapsulation primitives | [SWS_CRYPT_03000]<br>[SWS_CRYPT_03002]<br>[SWS_CRYPT_03003]<br>[SWS_CRYPT_03004]<br>[SWS_CRYPT_03005]<br>[SWS_CRYPT_03006]<br>[SWS_CRYPT_03007]<br>[SWS_CRYPT_03008]<br>[SWS_CRYPT_03009]<br>[SWS_CRYPT_10042]<br>[SWS_CRYPT_20752]<br>[SWS_CRYPT_20753]<br>[SWS_CRYPT_21400]<br>[SWS_CRYPT_21800]<br>[SWS_CRYPT_21801]<br>[SWS_CRYPT_40967]<br>[SWS_CRYPT_40968] |
| **[RS_CRYPTO_-02301]** | The Crypto Stack API shall provide a standardized header files structure | [SWS_CRYPT_20099]<br>[SWS_CRYPT_30099]<br>[SWS_CRYPT_40099] |
| **[RS_CRYPTO_-02302]** | The Crypto Stack API shall support a streaming approach | [SWS_CRYPT_10701]<br>[SWS_CRYPT_10710]<br>[SWS_CRYPT_10750]<br>[SWS_CRYPT_10751]<br>[SWS_CRYPT_10752]<br>[SWS_CRYPT_10753]<br>[SWS_CRYPT_20312]<br>[SWS_CRYPT_20313]<br>[SWS_CRYPT_20314]<br>[SWS_CRYPT_21110]<br>[SWS_CRYPT_21111]<br>[SWS_CRYPT_21112]<br>[SWS_CRYPT_21113]<br>[SWS_CRYPT_21114]<br>[SWS_CRYPT_21115]<br>[SWS_CRYPT_21118]<br>[SWS_CRYPT_22110]<br>[SWS_CRYPT_22111]<br>[SWS_CRYPT_22112]<br>[SWS_CRYPT_22113]<br>[SWS_CRYPT_22114]<br>[SWS_CRYPT_22115]<br>[SWS_CRYPT_23614]<br>[SWS_CRYPT_23615] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_23616] |
| | | [SWS_CRYPT_23617] |
| | | [SWS_CRYPT_23618] |
| | | [SWS_CRYPT_23619] |
| | | [SWS_CRYPT_23620] |
| | | [SWS_CRYPT_23621] |
| | | [SWS_CRYPT_23622] |
| | | [SWS_CRYPT_23625] |
| | | [SWS_CRYPT_23626] |
| | | [SWS_CRYPT_23634] |
| | | [SWS_CRYPT_23635] |
| | | [SWS_CRYPT_23715] |
| | | [SWS_CRYPT_24714] |
| | | [SWS_CRYPT_24715] |
| [RS_CRYPTO_-02304] | The Crypto Stack API should support the possibility to move a state of a "counter mode" stream cipher to a random position | [SWS_CRYPT_23613] |
| [RS_CRYPTO_-02305] | The Crypto Stack design shall separate cryptographic API from key access API | [SWS_CRYPT_00004]<br>[SWS_CRYPT_00006]<br>[SWS_CRYPT_10000]<br>[SWS_CRYPT_20700]<br>[SWS_CRYPT_30100] |
| [RS_CRYPTO_-02306] | The Crypto Stack shall support integration with a Public Key Infrastructure (PKI) | [SWS_CRYPT_20001]<br>[SWS_CRYPT_20002]<br>[SWS_CRYPT_20003]<br>[SWS_CRYPT_20004]<br>[SWS_CRYPT_20005]<br>[SWS_CRYPT_20006]<br>[SWS_CRYPT_20007]<br>[SWS_CRYPT_20009]<br>[SWS_CRYPT_20010]<br>[SWS_CRYPT_20011]<br>[SWS_CRYPT_20301]<br>[SWS_CRYPT_20302]<br>[SWS_CRYPT_20303]<br>[SWS_CRYPT_20304]<br>[SWS_CRYPT_20601]<br>[SWS_CRYPT_20602]<br>[SWS_CRYPT_20603]<br>[SWS_CRYPT_20611]<br>[SWS_CRYPT_20612]<br>[SWS_CRYPT_20613]<br>[SWS_CRYPT_20614]<br>[SWS_CRYPT_20615]<br>[SWS_CRYPT_20616]<br>[SWS_CRYPT_20617] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_20618] |
| | | [SWS_CRYPT_20619] |
| | | [SWS_CRYPT_20901] |
| | | [SWS_CRYPT_20902] |
| | | [SWS_CRYPT_20903] |
| | | [SWS_CRYPT_20904] |
| | | [SWS_CRYPT_20905] |
| | | [SWS_CRYPT_20906] |
| | | [SWS_CRYPT_20907] |
| | | [SWS_CRYPT_20908] |
| | | [SWS_CRYPT_20909] |
| | | [SWS_CRYPT_20910] |
| | | [SWS_CRYPT_22501] |
| | | [SWS_CRYPT_22503] |
| | | [SWS_CRYPT_24414] |
| | | [SWS_CRYPT_24415] |
| | | [SWS_CRYPT_40001] |
| | | [SWS_CRYPT_40002] |
| | | [SWS_CRYPT_40099] |
| | | [SWS_CRYPT_40100] |
| | | [SWS_CRYPT_40101] |
| | | [SWS_CRYPT_40111] |
| | | [SWS_CRYPT_40112] |
| | | [SWS_CRYPT_40113] |
| | | [SWS_CRYPT_40114] |
| | | [SWS_CRYPT_40115] |
| | | [SWS_CRYPT_40150] |
| | | [SWS_CRYPT_40151] |
| | | [SWS_CRYPT_40152] |
| | | [SWS_CRYPT_40153] |
| | | [SWS_CRYPT_40154] |
| | | [SWS_CRYPT_40155] |
| | | [SWS_CRYPT_40156] |
| | | [SWS_CRYPT_40157] |
| | | [SWS_CRYPT_40158] |
| | | [SWS_CRYPT_40159] |
| | | [SWS_CRYPT_40200] |
| | | [SWS_CRYPT_40201] |
| | | [SWS_CRYPT_40202] |
| | | [SWS_CRYPT_40203] |
| | | [SWS_CRYPT_40211] |
| | | [SWS_CRYPT_40212] |
| | | [SWS_CRYPT_40213] |
| | | [SWS_CRYPT_40214] |
| | | [SWS_CRYPT_40215] |
| | | [SWS_CRYPT_40216] |
| | | [SWS_CRYPT_40217] |
| | | [SWS_CRYPT_40218] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_40219] |
| | | [SWS_CRYPT_40220] |
| | | [SWS_CRYPT_40221] |
| | | [SWS_CRYPT_40300] |
| | | [SWS_CRYPT_40301] |
| | | [SWS_CRYPT_40302] |
| | | [SWS_CRYPT_40311] |
| | | [SWS_CRYPT_40313] |
| | | [SWS_CRYPT_40314] |
| | | [SWS_CRYPT_40315] |
| | | [SWS_CRYPT_40400] |
| | | [SWS_CRYPT_40401] |
| | | [SWS_CRYPT_40402] |
| | | [SWS_CRYPT_40403] |
| | | [SWS_CRYPT_40411] |
| | | [SWS_CRYPT_40412] |
| | | [SWS_CRYPT_40413] |
| | | [SWS_CRYPT_40414] |
| | | [SWS_CRYPT_40415] |
| | | [SWS_CRYPT_40416] |
| | | [SWS_CRYPT_40417] |
| | | [SWS_CRYPT_40418] |
| | | [SWS_CRYPT_40500] |
| | | [SWS_CRYPT_40501] |
| | | [SWS_CRYPT_40511] |
| | | [SWS_CRYPT_40600] |
| | | [SWS_CRYPT_40601] |
| | | [SWS_CRYPT_40602] |
| | | [SWS_CRYPT_40603] |
| | | [SWS_CRYPT_40604] |
| | | [SWS_CRYPT_40611] |
| | | [SWS_CRYPT_40612] |
| | | [SWS_CRYPT_40613] |
| | | [SWS_CRYPT_40614] |
| | | [SWS_CRYPT_40615] |
| | | [SWS_CRYPT_40616] |
| | | [SWS_CRYPT_40617] |
| | | [SWS_CRYPT_40618] |
| | | [SWS_CRYPT_40619] |
| | | [SWS_CRYPT_40620] |
| | | [SWS_CRYPT_40621] |
| | | [SWS_CRYPT_40622] |
| | | [SWS_CRYPT_40624] |
| | | [SWS_CRYPT_40625] |
| | | [SWS_CRYPT_40626] |
| | | [SWS_CRYPT_40627] |
| | | [SWS_CRYPT_40628] |
| | | [SWS_CRYPT_40629] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_40630] |
| | | [SWS_CRYPT_40631] |
| | | [SWS_CRYPT_40632] |
| | | [SWS_CRYPT_40633] |
| | | [SWS_CRYPT_40634] |
| | | [SWS_CRYPT_40635] |
| | | [SWS_CRYPT_40636] |
| | | [SWS_CRYPT_40637] |
| | | [SWS_CRYPT_40638] |
| | | [SWS_CRYPT_40639] |
| | | [SWS_CRYPT_40640] |
| | | [SWS_CRYPT_40641] |
| | | [SWS_CRYPT_40700] |
| | | [SWS_CRYPT_40701] |
| | | [SWS_CRYPT_40702] |
| | | [SWS_CRYPT_40711] |
| | | [SWS_CRYPT_40800] |
| | | [SWS_CRYPT_40801] |
| | | [SWS_CRYPT_40802] |
| | | [SWS_CRYPT_40811] |
| | | [SWS_CRYPT_40900] |
| | | [SWS_CRYPT_40912] |
| | | [SWS_CRYPT_40913] |
| | | [SWS_CRYPT_40914] |
| | | [SWS_CRYPT_40915] |
| | | [SWS_CRYPT_40916] |
| | | [SWS_CRYPT_40917] |
| | | [SWS_CRYPT_40918] |
| | | [SWS_CRYPT_40919] |
| | | [SWS_CRYPT_40920] |
| | | [SWS_CRYPT_40921] |
| | | [SWS_CRYPT_40922] |
| | | [SWS_CRYPT_40923] |
| | | [SWS_CRYPT_40924] |
| | | [SWS_CRYPT_40925] |
| | | [SWS_CRYPT_40926] |
| | | [SWS_CRYPT_40927] |
| | | [SWS_CRYPT_40928] |
| | | [SWS_CRYPT_40929] |
| | | [SWS_CRYPT_40930] |
| | | [SWS_CRYPT_40931] |
| | | [SWS_CRYPT_40932] |
| | | [SWS_CRYPT_40933] |
| | | [SWS_CRYPT_40934] |
| | | [SWS_CRYPT_40935] |
| | | [SWS_CRYPT_40936] |
| | | [SWS_CRYPT_40937] |
| | | [SWS_CRYPT_40938] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_40939] |
| | | [SWS_CRYPT_40940] |
| | | [SWS_CRYPT_40941] |
| | | [SWS_CRYPT_40942] |
| | | [SWS_CRYPT_40943] |
| | | [SWS_CRYPT_40972] |
| | | [SWS_CRYPT_40973] |
| | | [SWS_CRYPT_40974] |
| | | [SWS_CRYPT_40975] |
| | | [SWS_CRYPT_40976] |
| | | [SWS_CRYPT_40977] |
| | | [SWS_CRYPT_40978] |
| | | [SWS_CRYPT_40979] |
| | | [SWS_CRYPT_40980] |
| | | [SWS_CRYPT_40981] |
| [RS_CRYPTO_-02307] | The Crypto Stack design shall separate cryptographic API from the PKI API | [SWS_CRYPT_20000] |
| | | [SWS_CRYPT_20700] |
| | | [SWS_CRYPT_24400] |
| | | [SWS_CRYPT_24401] |
| | | [SWS_CRYPT_24410] |
| [RS_CRYPTO_-02308] | The Crypto Stack shall support a unified cryptographic primitives naming convention, common for all suppliers | [SWS_CRYPT_03904] |
| | | [SWS_CRYPT_03905] |
| | | [SWS_CRYPT_03906] |
| | | [SWS_CRYPT_03910] |
| | | [SWS_CRYPT_20651] |
| | | [SWS_CRYPT_20711] |
| | | [SWS_CRYPT_20712] |
| | | [SWS_CRYPT_40970] |
| | | [SWS_CRYPT_40971] |
| [RS_CRYPTO_-02309] | The Crypto Stack API shall support the run-time configurable usage style | [SWS_CRYPT_20103] |
| | | [SWS_CRYPT_20412] |
| | | [SWS_CRYPT_20516] |
| | | [SWS_CRYPT_20652] |
| | | [SWS_CRYPT_21415] |
| | | [SWS_CRYPT_21416] |
| | | [SWS_CRYPT_21514] |
| | | [SWS_CRYPT_21715] |
| | | [SWS_CRYPT_21817] |
| | | [SWS_CRYPT_21818] |
| | | [SWS_CRYPT_22213] |
| | | [SWS_CRYPT_22214] |
| | | [SWS_CRYPT_23213] |
| | | [SWS_CRYPT_23214] |
| | | [SWS_CRYPT_23312] |
| | | [SWS_CRYPT_23611] |
| | | [SWS_CRYPT_23612] |
| | | [SWS_CRYPT_23624] |
| | | [SWS_CRYPT_23711] |
| | | [SWS_CRYPT_23712] |
| | | [SWS_CRYPT_24411] |
| | | [SWS_CRYPT_24412] |
| | | [SWS_CRYPT_24413] |
| | | [SWS_CRYPT_29000] |

| Requirement | Description | Satisfied by |
|---|---|---|
| | | [SWS_CRYPT_29001] |
| | | [SWS_CRYPT_29002] |
| | | [SWS_CRYPT_29003] |
| | | [SWS_CRYPT_29004] |
| | | [SWS_CRYPT_29010] |
| | | [SWS_CRYPT_29011] |
| | | [SWS_CRYPT_29012] |
| | | [SWS_CRYPT_29013] |
| | | [SWS_CRYPT_29014] |
| | | [SWS_CRYPT_29015] |
| | | [SWS_CRYPT_29020] |
| | | [SWS_CRYPT_29021] |
| | | [SWS_CRYPT_29022] |
| | | [SWS_CRYPT_29023] |
| | | [SWS_CRYPT_29024] |
| | | [SWS_CRYPT_29030] |
| | | [SWS_CRYPT_29031] |
| | | [SWS_CRYPT_29032] |
| | | [SWS_CRYPT_29033] |
| | | [SWS_CRYPT_29034] |
| | | [SWS_CRYPT_29035] |
| | | [SWS_CRYPT_29040] |
| | | [SWS_CRYPT_29041] |
| | | [SWS_CRYPT_29042] |
| | | [SWS_CRYPT_29043] |
| | | [SWS_CRYPT_29044] |
| | | [SWS_CRYPT_29045] |
| | | [SWS_CRYPT_29047] |
| | | [SWS_CRYPT_29048] |
| | | [SWS_CRYPT_29049] |
| | | [SWS_CRYPT_40984] |
| | | [SWS_CRYPT_40985] |
| [RS_CRYPTO_-02310] | The Crypto Stack API shall support an efficient mechanism of error states notification | [SWS_CRYPT_00104] |
| | | [SWS_CRYPT_10099] |
| | | [SWS_CRYPT_19902] |
| | | [SWS_CRYPT_19903] |
| | | [SWS_CRYPT_19904] |
| | | [SWS_CRYPT_19905] |
| | | [SWS_CRYPT_19906] |
| | | [SWS_CRYPT_19950] |
| | | [SWS_CRYPT_19951] |
| | | [SWS_CRYPT_19953] |
| | | [SWS_CRYPT_19954] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[RS_CRYPTO_-02401]** | The Crypto Stack should support a joint usage of multiple back-end cryptography providers including ones with non-extractable keys | [SWS_CRYPT_00005]<br>[SWS_CRYPT_00006]<br>[SWS_CRYPT_00009]<br>[SWS_CRYPT_10017]<br>[SWS_CRYPT_20098]<br>[SWS_CRYPT_20099]<br>[SWS_CRYPT_20654]<br>[SWS_CRYPT_20700]<br>[SWS_CRYPT_30001]<br>[SWS_CRYPT_30002]<br>[SWS_CRYPT_30003]<br>[SWS_CRYPT_30099]<br>[SWS_CRYPT_30100]<br>[SWS_CRYPT_30130]<br>[SWS_CRYPT_30131]<br>[SWS_CRYPT_30403]<br>[SWS_CRYPT_40911] |
| **[RS_CRYPTO_-02403]** | The Crypto Stack shall support isolating keys and requests | [SWS_CRYPT_22500]<br>[SWS_CRYPT_23800]<br>[SWS_CRYPT_24802] |
| **[RS_CRYPTO_-02405]** | The Crypto Stack shall support the key slots identification in a way independent from a concrete deployment | [SWS_CRYPT_10005]<br>[SWS_CRYPT_30400]<br>[SWS_CRYPT_30401]<br>[SWS_CRYPT_30402] |

Document ID 883: AUTOSAR_SWS_Cryptography

# 7 Functional specification

The AUTOSAR Adaptive architecture organizes the software of the AUTOSAR Adaptive foundation as `Functional Clusters`. These clusters offer common functionality as services to the applications. The `Functional Cluster` Cryptography (`FC Crypto`) is part of the AUTOSAR Adaptive Platform Foundation.

The `FC Crypto` provides the infrastructure to access multiple implementations of cryptographic operations through a standardized interface, `CryptoAPI`. Operations provided by `FC Crypto` are grouped into different *providers*, each of them implements specific domain of cryptography-related functionality:

- `Crypto Provider`
- `Key Storage Provider`
- `X.509` Certificate Management Provider

This specification includes the syntax of the `API`, the relationship of the `API` to the model and describes semantics.

## 7.1 Functional Cluster Lifecycle

### 7.1.1 Startup

Using `ara::core::Intitialize` and `ara::core::Deinitialize`, the application can initialize and deinitialize `FC Crypto` resources allocated to the application.

**[SWS_CRYPT_00101]**{DRAFT} ⌈When `ara::core::Intitialize` is called, the `FC Crypto` shall read in the manifest information and prepare the access structures to `CryptoProvider` and `CryptoKeySlot` that are defined in the manifest.

⌋*(RS_CRYPTO_02110)*
Hint: Access structures may encompass the communication channel between the application process and the stack process or other resource required by the `CryptoAPI`.

### 7.1.2 Shutdown

**[SWS_CRYPT_00102]**{DRAFT} ⌈When `ara::core::Deinitialize` is called, the `FC Crypto` shall ensure that all open contexts are closed and all occupied ressources are freed.⌋*(RS_CRYPTO_02004, RS_CRYPTO_02007, RS_CRYPTO_02102)*

`ara::crypto::cryp::CryptoObject::CryptoObject`,    `ara::crypto::cryp::CryptoContext`

**[SWS_CRYPT_00103]**{DRAFT} ⌈When `ara::core::Deinitialize` is called, the `FC Crypto` shall ensure that all associated persist operations in this context of this

application are executed successfully and no new persist operations are started.⌋*(RS_-CRYPTO_02004)*

Note: the application is expected not to call any API of `FC Crypto` before `ara::-core::Intitialize` or after `ara::core::Deinitialize`.

**[SWS_CRYPT_00104]**{DRAFT} ⌈All functions of `FC Crypto` and all methods of its classes shall return the error `kUnknownIdentifier` when they are called after static initialization but before `ara::core::Intitialize` was called or after `ara:-:core::Deinitialize` was called.⌋*(RS_CRYPTO_02310)*

## 7.2 Architectural concepts

The `FC Crypto` offers applications and other Adaptive AUTOSAR `Functional Clusters` a standardized interface, which provides operations for cryptographic and related calculations. These operations include cryptographic operations, key management and certificate handling. `FC Crypto` handles the actual implementation of all operations, including all necessary configuration and brokering of operations between requesting-application and `FC Crypto`-provided implementation. The standardized interface is exposed by the `CryptoAPI`.
The `FC Crypto` and its `CryptoAPI` support both public-key and symmetric-key cryptography. It allows applications to use mechanisms such as authentication, encryption and decryption for automotive services.
The interfaces defined by `FC Crypto` are designed to enable integraton of 3rd party cryptographic libraries and hardware-based elements. This facilitates implementation of a security "trust anchor" or acceleration of cryptographic transformations in situations, where the `FC Crypto`"s default crypto-library will not provide the necessary primitives or hardware acceleration is needed.

`CryptoAPI` provides a set of methods, which enable application and system developer to store and transmit information while safeguarding it from intruders. `CryptoAPI` provides cryptographic methods to keep critical information in confidential and / or authentic form, and to communicate in a way such that only the intended recipient can read the message. Therefore, `FC Crypto` provides mechanisms for building applications that ensure the following security goals:

- Authentication: `FC Crypto` provides mechanisms that allow `Adaptive Applications` or `Functional Clusters` to prove their identity to other applications or `Functional Clusters`.

- Non-Repudiation: `FC Crypto` supports the concept of non-repudiation, where someone cannot deny the validity of something.

- Confidentiality: `FC Crypto` allows to keep information private. Cryptographic systems were originally developed to function in this capacity. Whether it be system or user specific data sent during system debugging or tracing, or storing

confidential vehicle / `ECU` data, encryption can assure that only users who have access to the appropriate key will get read access to the data `Plaintext`.

- Integrity: `FC Crypto` ensures that secured data is not altered during storage or transmission without the receiver detecting this altering. Additionally, `FC Crypto` allows applications to build functionality, which guarantees the integrity of elements or services.

The `FC Crypto` shall take care not to leak any information about the message it has read from a stream, until the decryption process has finished without error.

Additionally, the `FC Crypto` integrates a `Key Storage Provider`. The purpose of this element is secure persistent storage of any supported cryptographic objects and programmatic access to them via a unified interface, independently from actual physical storage implementations. A single logical Key Storage can aggregate multiple software or hardware-based physical storage managed by the correspondent `Crypto Providers`. This is done transparent for the user of the Key Storage interface. Guaranteeing correct access to the keys, `CryptoAPI` restricts access to this material.

`CryptoAPI` allows to manage `PKI` certificates. These interfaces are grouped in a certificate management namespace. Here, all typical certificate handling mechanism, such as issuing, revocation, and replacement, are handled. Additionally, certificate management `API` provides a kind of permanent storage where all certificates are stored. All operations on certificates are done by certificate management, which enforces access permissions by implementing the `Policy Enforcement Point`.

The definition and implementation of `FC Crypto` shall be implemented according to its parts as described above. The architectural overview shows all parts, such as `X.509 Provider` for certificate handling, `Crypto Provider` and `Key Storage Provider`. Figure 7.1 depicts the high-level architecture of `FC Crypto` including the previously described elements.
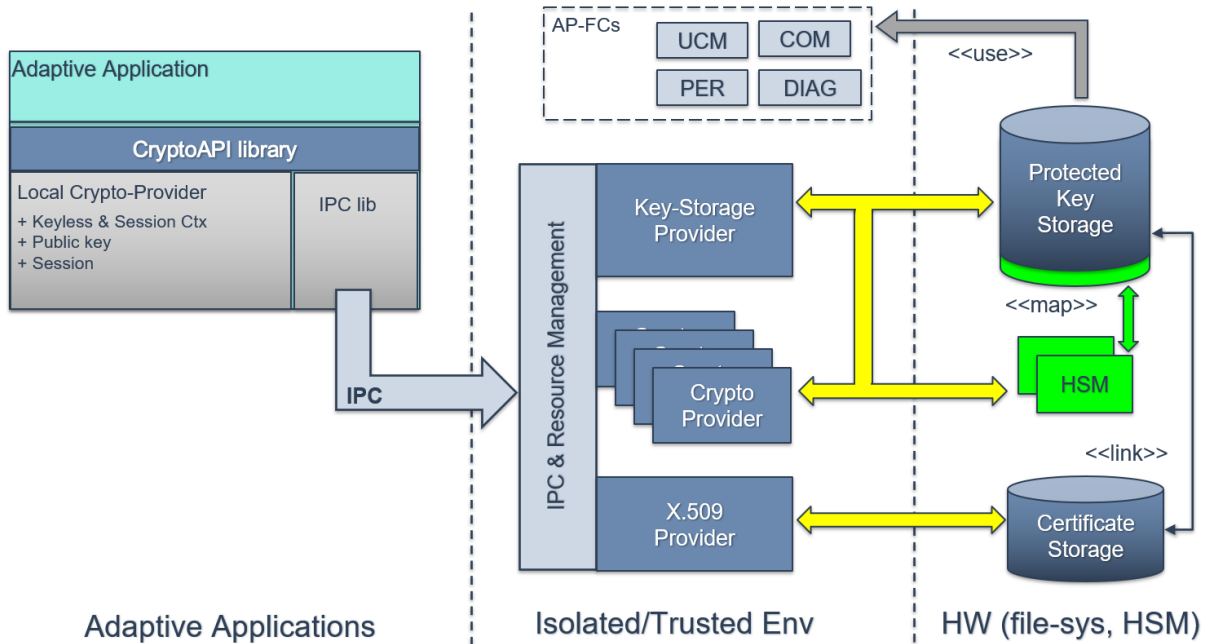
**Figure 7.1: High-level `CryptoAPI` architecture**

### 7.2.1 Integration with Identity and Access Management

To enable access control the `FC Crypto` shall implement a `Policy Enforcement Point` (`PEP`) to enforce the policy decision obtained from the `Policy Decision Point` (`PDP`) as specified by `Identity and Access Management` (`IAM`). Thus, an interaction is needed between `FC Crypto` (`PEP`) and some entity that implements the `PDP`.

Since only key- and certificate-slots are subject to access control it makes sense to embed the `PEP` within the `Key Storage Provider` and the `X.509 Provider`. One possible implementation is illustrated in figure 7.2: a `PDP` interface (`IAM` unit) obtains policy information and decides whether access is granted; this decision is enforced by a `PEP` functional unit. Both units may be implemented as part of the `Key Storage Provider`.
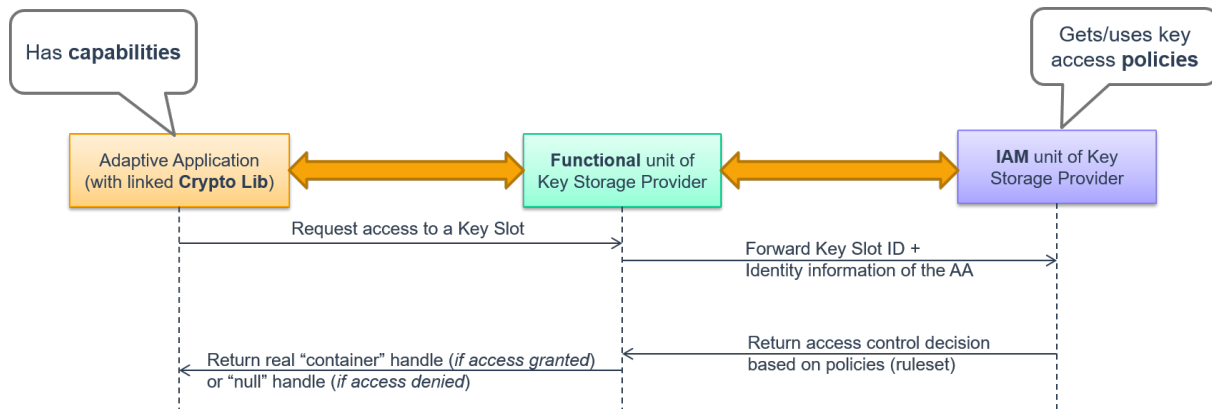


**Figure 7.2: Interaction with `IAM`**

`IAM` enables access control to modeled entities or resources. Currently, `FC Crypto` considers access control only for two types of resources: `Key Slot` (read/write) and `Certificate Slot` (write). To simplify IAM configuration `FC Crypto` specifies the exclusive-access-model, which states that access to a key-slot can only be granted to a single `Adaptive Application` (exclusive).

Clarification: key-slots and certificate-slots are non-volatile in nature, i.e. there is no use case for allocating volatile key-slot or certificate-slot instances.

Note: `Functional Cluster` access to a `Key Slot` assigned under exlusive-access to an `Adaptive Application` is not ruled out by this model (see sub-chapter 7.2.2)!

To enable and synchronize concurrent update and usage of the same key-slot, the `Key Storage Provider` specifies dedicated interfaces and mechanisms, which are subject to access control based on the addressed `Key Slot`. Figure 7.3 showcases this scenario: the `Adaptive Application` has exclusive-access to a `Key Slot`, which is used by a library providing cryptographic services to a higher layer (business logic). At the same time another library independently manages `Key Slot` content (e.g. crypto-keys).
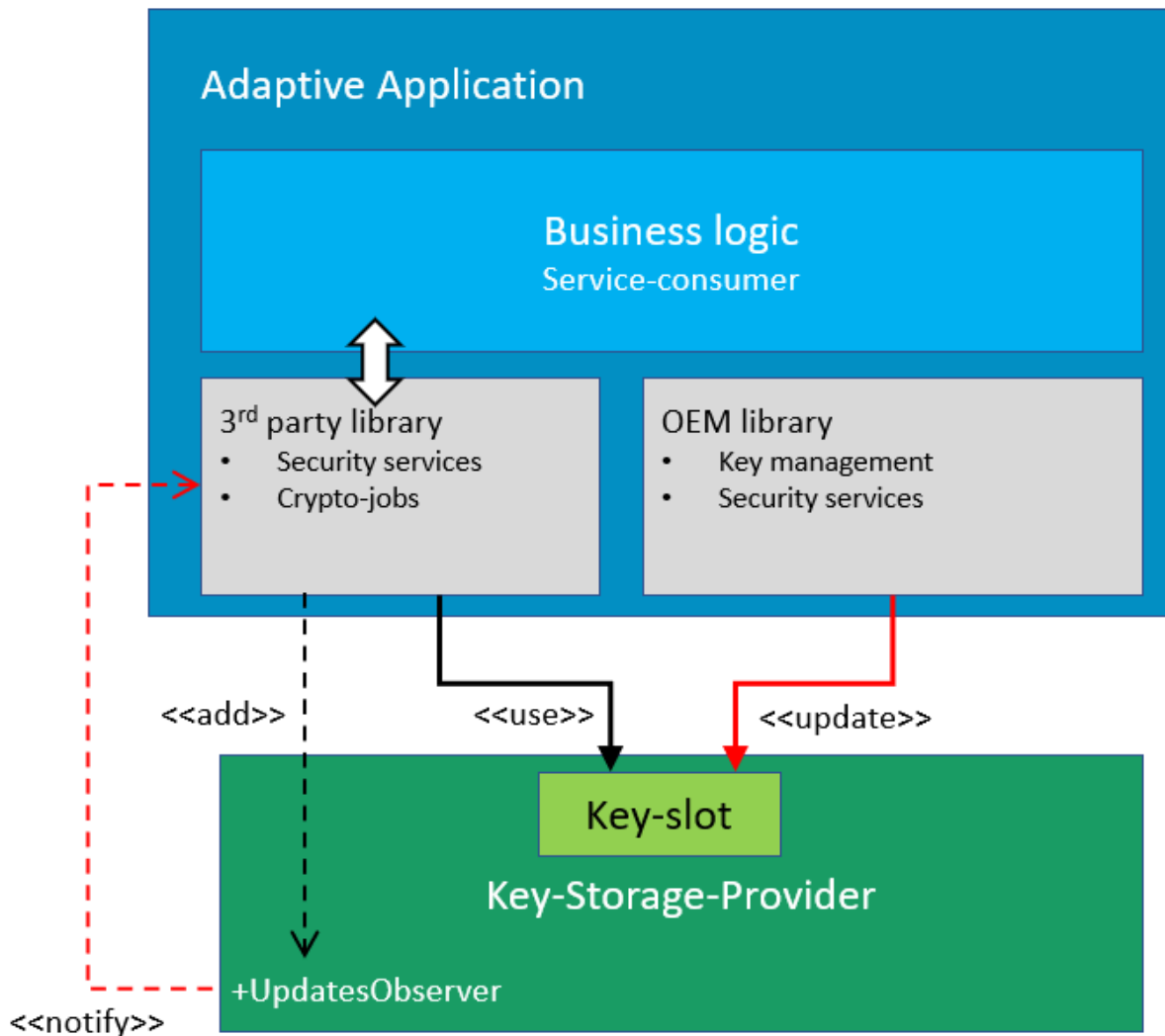
**Figure 7.3: Concurrent access to a single Key Slot**

The required `Key Slots` are described in the manifest of the application. This information is stored by `IAM`, e.g. in a database.

### 7.2.2 Integration into AUTOSAR

The overall architecture is described in chapter 7.2. The `FC Crypto` provides its service to all AUTOSAR elements, such as untrusted `Adaptive Applications` or trusted system services (`Functional Clusters`). From cryptographic service point of view both could be treated equally.

The integraton of FC Crypto into AUTOSAR is described in Figure 7.4.



**Figure 7.4: Integration into AUTOSAR**

Their differential treatment is due to the underlying trust-model: system services (`Functional Clusters`) are the trusted foundation while `Adaptive Applications` are untrusted additions. To ensure secure access from application side the trust-model, in the form of IAM, is designed for and applied only to `Adaptive Applications`. Similarly, the exclusive-access-model aims at protecting an application's own resources against access by other applications, but additionally also against access by `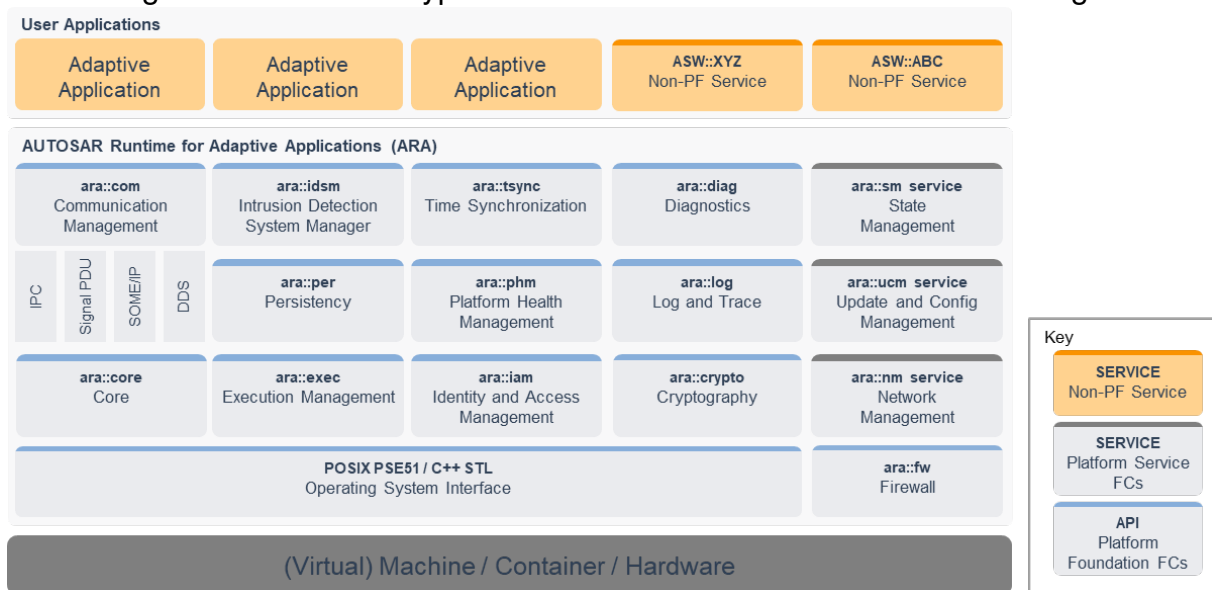Functional Clusters` other than `FC Crypto`. On the other hand some `Functional Clusters` specify their own key-slots, which contain key-material to be used when implementing certain system services (e.g. secure data storage, secure diagnostics or secure communication such as SecOC). Because key-management of `Key Slots` used by `Functional Clusters` should be possible from an `Adaptive Application` (e.g. OEM key manager), the exclusive-access-model defines two types of `Key Slots`:

- **application**: the application has exclusive access to this key slot. It is able to import/export, update/delete and use the contained key-material. No other application or `Functional Cluster` may access this `Key Slot`.

- **machine**: this type of `Key Slot` is defined by the adaptive machine and may be used by the `Functional Cluster` for which it is configured. Additionally, the `Key Slot` may be assigned to a single `Adaptive Application` that is then able to manage the contained key-material.

Figure 7.5 gives an example for the use of machine and application `Key Slots`.
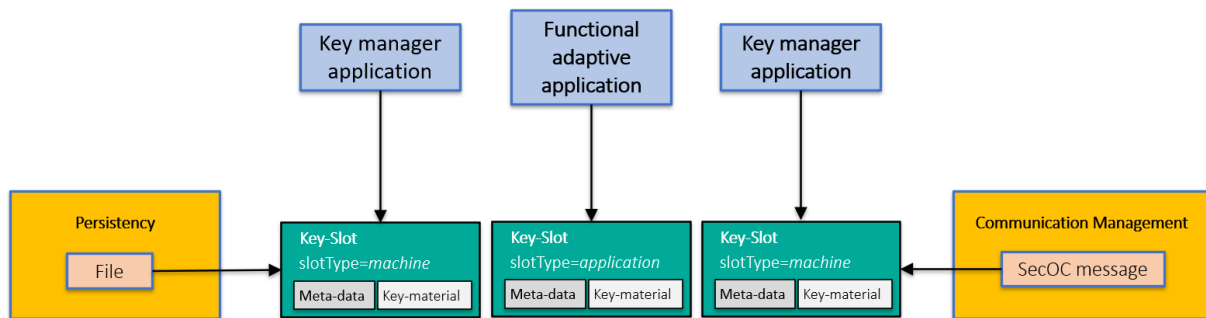
**Figure 7.5: `Key Slot` types and usages**

### 7.2.3 Application level

The `FC Crypto` has been primarily designed to enable `Adaptive Applications` to access cryptographic services, for a majority of which cryptographic key-material is needed. Therefore, an application may define the required `Key Slots`, `Crypto Providers` and certificates. These information are represented in the design model. The `CryptoKeySlotInterface` describes the needed `Key Material` for an application.

When a specified `Key Slot` is of slotType *application*, the application expects **exclusive** access to this key-slot. During Integration a key-slot resource must be allocated on the machine.

When an `Adaptive Application` specifies a `Key Slot` of slotType *machine*, it expresses a wish to **manage** a platform `Key Slot` with the configured properties. Note: the attribute cryptoKeyName of `CryptoKeySlotInterface` is used to match platform `Key Slots` and application-manifest specified *machine* `Key Slots`.

An `Adaptive Application` that uses a `Crypto Provider` without keys (e.g. Hashing, Random Number Generation) or only session keys may use the `CryptoProviderInterface`. Additionally, if the application requires certificates, this can be configured using the `CryptoCertificateInterface`. Figure 7.6 shows the model elements that are used to configure access from an `Adaptive Application` to elements of `FC Crypto`.
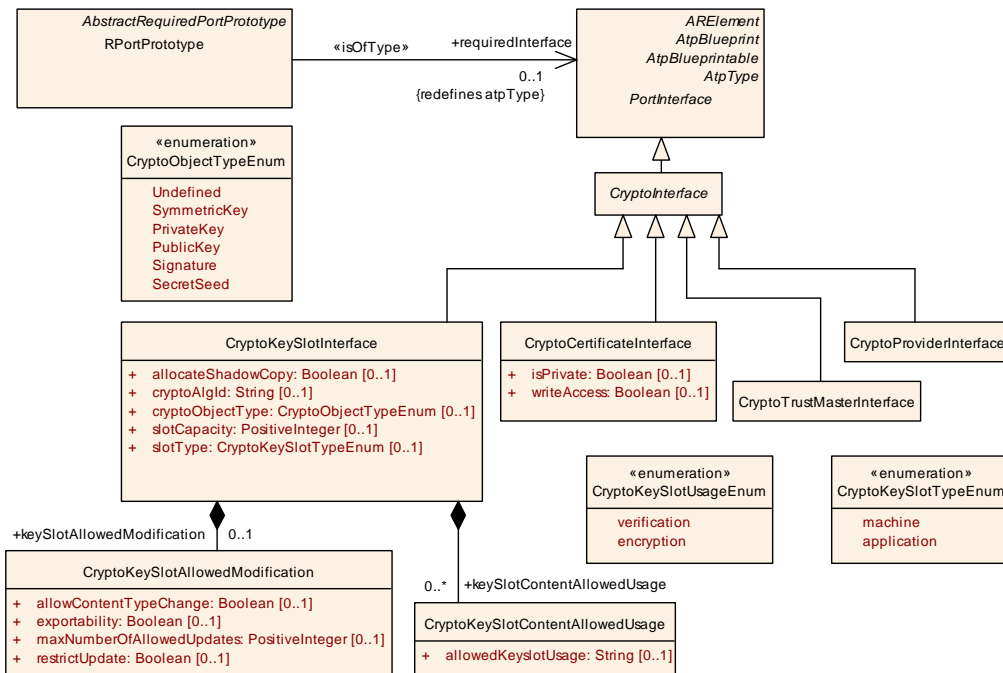
**Figure 7.6: Application interface**

### 7.2.4 System service level

Some `Adaptive Platform Services` such as update and configuration, communication, persistency or diagnostics also require cryptographic services as part of their functionality. If key-material is needed and must be configurable by an `Adaptive Application` (e.g. OEM key manager), the platform shall specify a `Key Slot` of slotType *machine*. To manage the `Key Material` a dedicated `Adaptive Application` (key-manager) may specify the same `Key Slot` (i.e. same parameters and slotType *machine*). During Integration this machine type key-slot resource must be linked to the key-manager.

### 7.2.5 Bridging domains: the IOInterface

One major design decision of `FC Crypto` is to separate to the extent possible the three domains dealing with cryptography (crypto::cryp), key management (crypto::keys) and certificate management (crypto::x509). To simplify interaction between domains and abstract interfaces from the actual object the IOInterface interface has been introduced as an intermediate layer between the persistent resource and the runtime object. The IOInterface represents a smart wrapper providing access to and meta-data on the content it is encapsulating. For example, it can be used by an application to instantiate a runtime crypto-object from its persistent storage location (read-access). Or it can be used by an application to store a runtime crypto-object into a persistent storage location (write-access).

## 7.3 Crypto API structure

CryptoAPI provided by FC Crypto to consumers is presented by three different Provider types, each of them implements specific domain of cryptography-related functionality:

1. **Crypto Provider** (CP, namespace `ara::crypto::cryp`) is responsible for implementation of all supported Cryptographic primitives. FC Crypto may support multiple instances of the Crypto Providers. Each instance of Crypto Provider represents single holistic software- or hardware-based implementation of some set of cryptographic algorithms. Each Crypto Provider must isolate all Key Material used during processing from unauthorized access from "external world".

2. **Key Storage Provider** (KSP, namespace `ara::crypto::keys`) is responsible for secure (confidential and/or authentic) storage of different type Key Material (public/private/secret keys, seeds) and other security critical cryptographic objects (digital signatures, hash, MAC/HMAC tags). CryptoAPI consumers work with logically single KSP that is used for access to all crypto objects independently from their physical hosting on the ECU. But from the stack supplier point of view, each HSM may support own back-end KSP responsible for access control to internally stored cryptographic objects. All back-end KSP are hidden from the consumers (under public CryptoAPI). KSP implementation (similar to Crypto Provider) must ensure confidentiality and authenticity of processed and stored objects, i.e. its implementation must be isolated from the consumers' code space.

3. **X.509 Certificate Management Provider** (CMP, namespace `ara::crypto::x509`) is responsible for X.509 certificates parsing, verification, authentic storage and local searching by different attributes. Also CMP is responsible for storage, management and processing of Certificate Revocation Lists (CRLs) and Delta CRLs. CMP supports of requests preparation and responses parsing for On-line Certificate Status Protocol (OCSP). FC Crypto supports only single instance of the CMP and it is completely independent from Crypto Provider and KSP implementation details, therefore CMP and Crypto Provider/KSP may be provided by completely independent suppliers. **Note:** CMP works with non-confidential objects only.

**Note:** Public APIs of each Provider type is common for consumers code and components suppliers. It is a mandatory part of API. But Crypto Provider and back-end KSP from single supplier may use internal "private" APIs for intercommunication. Also FC Crypto may specify additional "protected" APIs expected from specific provider type.

## 7.4 Crypto API elements

### 7.4.1 Crypto Provider

A `Crypto Provider` is a structural element that organizes `Cryptographic primitives`. Every `Crypto Provider` represents exactly either one hardware element, e.g., trusted platform module (TPM) or hardware security module (HSM), or one software element, e.g., cryptographic library. As a general rule, the stack vendor is expected to provide at least one `Crypto Provider` for each hardware and/or software element that is available in a project specific environment.

**[SWS_CRYPT_00004]**{DRAFT} ⌈Each derived implementation of the interface class `ara::crypto::cryp::CryptoProvider` shall encapsulate cryptographic transformations and associated resources, such as `ara::crypto::cryp::CryptoObject` and `Cryptographic primitives`, of a single software or hardware cryptography implementation.⌋*(RS_CRYPTO_02305)*

Note: a `Crypto Provider` may expose only a subset of all available transformations or primitives of the underlying software or hardware cryptography implementation (e.g. in case of weak or outdated primitives). However, this implementation detail shall be documented and communicated to the user.

**[SWS_CRYPT_00005]**{DRAFT} ⌈The global factory method `ara::crypto::LoadCryptoProvider` shall instantiate a `ara::crypto::cryp::CryptoProvider` identified by the provided `ara::core::InstanceSpecifier`.⌋*(RS_CRYPTO_02401)*

**[SWS_CRYPT_00006]**{DRAFT} ⌈Each instance of a `Crypto Provider` shall implement one coherent representation of either software based cryptographic algorithms, i.e. library, or hardware based cryptographic algorithms, e.g., `HSM`.⌋*(RS_CRYPTO_02305, RS_CRYPTO_02401)*

**[SWS_CRYPT_00007]**{DRAFT} ⌈Derived implementations of the interface class `ara::crypto::cryp::CryptoProvider` shall isolate all non-session `Key Material` from the user (`Adaptive Application`).⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002)*

**[SWS_CRYPT_00009]**{DRAFT} ⌈The `ara::crypto::cryp::CryptoProvider` shall be identified during runtime via call to `ara::crypto::LoadCryptoProvider` with `ara::core::InstanceSpecifier` as an input parameter. Here `ara::core::InstanceSpecifier` represents a path to `RPortPrototype` mapped to referenced `ara::crypto::cryp::CryptoProvider`.⌋*(RS_CRYPTO_02401)*

**[SWS_CRYPT_00500]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx` shall return an instance of `ara::crypto::cryp::RandomGeneratorCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The instantiated `ara::crypto::cryp::RandomGeneratorCtx` shall only be seeded, if a local-state

`ara::crypto::cryp::RandomGeneratorCtx` shall be created and the optional boolean parameter initialize is not provided or set to TRUE. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported or equals `kAlgIdDefault` but the `ara::crypto::cryp::CryptoProvider` does not support random number generation.

- `kBusyResource`, if seeding is requested but cannot be provided.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to random number generation.

⌋*(RS_CRYPTO_02206)*

While this enables applications to create a ready-to-go RandomGeneratorCtx, it cannot be guaranteed that seeding of the RandomGeneratorCtx is possible at this point in time, e.g., due to a lack of entropy. As applications shall be prevented from modifying the state of global-state RandomGeneratorCtx, applications shall also not be able to trigger the seeding of any global-state RandomGeneratorCtx.

**[SWS_CRYPT_00506]**{DRAFT} ⌈If `ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx` is called to create a global-state `ara::crypto::cryp::RandomGeneratorCtx`, the optional boolean parameter initialize shall be ignored and the requested RandomGeneratorCtx shall be returned without modification of its state.⌋*(RS_CRYPTO_02206)*

**[SWS_CRYPT_00601]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateKeyDerivationFunctionCtx` shall return an instance of `ara::crypto::cryp::KeyDerivationFunctionCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to key derivation.

⌋*(RS_CRYPTO_02101)*

This context needs an identifier to specify the used cryptographic algorithm. This identifier is encoded with the common name as defined in chapter 7.5. This context will also be used in different areas to derive keys, such as Key Agreement or Key Encapsulation.

**[SWS_CRYPT_00901]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateHashFunctionCtx` shall return an instance of `ara::crypto::cryp::HashFunctionCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to hashing.

⌋*(RS_CRYPTO_02205)*

The `ara::crypto::CryptoAlgId` identifier represents the common name as defined in chapter 7.5.

**[SWS_CRYPT_40960]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateHashDigest` shall return an instance of `ara::crypto::cryp::Signature` initialized according to the provided `ara::crypto::CryptoAlgId` and hash data (ara::crypto::ReadOnlyMemRegion value). The interface shall only support ara::crypto::CryptoAlgIds of hash primitives and return

- `kUnknownIdentifier` , if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument` , if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to hashing.

- `kInvalidInputSize` , if the size of hash data provided is not compatible with the provided `ara::crypto::CryptoAlgId`.

⌋*(RS_CRYPTO_02205)*

**[SWS_CRYPT_01200]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateMessageAuthnCodeCtx` shall return an instance of `ara::crypto::cryp::MessageAuthnCodeCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`.

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to `Message Authentication Code` generation or verification.

⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_40963]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateSymmetricBlockCipherCtx` shall return an instance of `ara::crypto::cryp::SymmetricBlockCipherCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to symmetric `Block Cipher` en/decryption.

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_40964]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateStreamCipherCtx` shall return an instance of `ara::-crypto::cryp::StreamCipherCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to symmetric `Stream Cipher` en/decryption

⌋*(RS_CRYPTO_02107, RS_CRYPTO_02201)*

**[SWS_CRYPT_01806]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateAuthCipherCtx` shall return an instance of `ara::crypto:-:cryp::AuthCipherCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to authenticated encryption/decryption.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_40965]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateSymmetricKeyWrapperCtx` shall return an instance of `ara:-:crypto::cryp::SymmetricKeyWrapperCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to symmetric key-wrapping.

⌋*(RS_CRYPTO_02208)*

**[SWS_CRYPT_02400]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateVerifierPublicCtx` shall return an instance of `ara::-crypto::cryp::VerifierPublicCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`.

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to signature verification.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02408]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateSignerPrivateCtx` shall return an instance of `ara::crypto::cryp::SignerPrivateCtx` implementing the primitive specified by the provided algorithm `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to signature generation..

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02409]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateSigEncodePrivateCtx` shall return an instance of `ara::crypto::cryp::SigEncodePrivateCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer signature generation with message encoding.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02410]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateMsgRecoveryPublicCtx` shall return an instance of ara::crypto::cryp:: `ara::crypto::cryp::MsgRecoveryPublicCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to signature verification with message recovery.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_40966]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateEncryptorPublicCtx` shall return an instance of `ara::crypto::cryp::EncryptorPublicCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer but does not refer to assymetric encryption.

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_40967]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateKeyEncapsulatorPublicCtx` shall return an instance of `ara::crypto::cryp::KeyEncapsulatorPublicCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to key encapsulation.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_40968]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateKeyDecapsulatorPrivateCtx` shall return an instance of `ara::crypto::cryp::KeyDecapsulatorPrivateCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to key decapsulation.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_40961]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateSignature` shall return an instance of `ara::crypto::cryp::Signature` initialized according to the provided primary `ara::crypto::CryptoAlgId`, authentication tag (ara::crypto::ReadOnlyMemRegion value), the provided `ara::crypto::cryp::RestrictedUseObject` key and optionally a secondary `ara::crypto::CryptoAlgId` of the hash function used with the primary digital signature algorithm. The interface shall only support primary ara::crypto::CryptoAlgIds of digital signature and `Message Authentication Code` primitives and return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided CryptoAlgIds are supported but do not refer to digital signature or `Message Authentication Code` primitives.

- `kInvalidInputSize`, if the size of the authentication tag provided is not compatible with the provided primary `ara::crypto::CryptoAlgId`.

- `kIncompatibleArguments`, if the primary and secondary `ara::crypto::CryptoAlgId`s are incompatible or if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::RestrictedUseObject` is incompatible with either the primary and secondary `ara::crypto::CryptoAlgId`.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_40962]**{DRAFT} ⌈The interfaces `ara::crypto::cryp::-CryptoProvider::GeneratePrivateKey`,`ara::crypto::cryp::Crypto-Provider::GenerateSeed` and `ara::crypto::cryp::CryptoProvider:-:GenerateSymmetricKey` shall generate secret key-material according to the provided `ara::crypto::CryptoAlgId` return an instance of `ara::crypto::-cryp::PrivateKey`, `ara::crypto::cryp::SecretSeed` or `ara::crypto::-cryp::SymmetricKey` respectively. Each function shall initialize the object according to the provided `ara::crypto::AllowedUsageFlags` and boolean attributes isSession and isExportable, e.g. as `isSession` or `isExportable`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to primitive that the interfaces shall generate.

⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_40969]**{DRAFT} ⌈The interface `ara::crypto::cryp::Crypto-Provider::CreateKeyAgreementPrivateCtx` shall return an instance of `ara:-:crypto::cryp::KeyAgreementPrivateCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer to key agreement.

⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_40970]**{DRAFT} **Translation of common name to vendor identifier** ⌈The interface `ara::crypto::cryp::CryptoProvider::ConvertToAlgId` shall convert the provided primitive name from its string representation according to NamingConvention into a vendor specific `ara::crypto::CryptoAlgId`. The interface shall return `kAlgIdUndefined`, if the provided primitive name is not supported.⌋ *(RS_CRYPTO_02308)*

**[SWS_CRYPT_40971]**{DRAFT} **Translation of identifier to name** ⌈The `ara::-crypto::cryp::CryptoProvider::ConvertToAlgName` shall convert a vendor specific algorithm identifier to the common name of the cryptographic algorithm.

The interface `ara::crypto::cryp::CryptoProvider::ConvertToAlgName` shall convert the provided vendor specific `ara::crypto::CryptoAlgId` into a primitive name according to NamingConvention. The interface shall return a ara::core::StringView of the converted primitive name or `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.⌋ *(RS_CRYPTO_02308)*

Note: generation of strong key-material is the foundation that underpins all security properties of further cryptographic transformations or protocols. It is the stack vendor's responsibility to ensure strong key-material is generated. The user of the above mentioned generate interfaces provides additional restrictions of how the generated key-material may be used, e.g. restricting usage of a `Symmetric Key` only to message authentication, forbidding the key-material to be exported or to be persistently stored (session keys).

The `ara::crypto::CryptoAlgId` is the implementation specific identifier that represents the algorithm name, as described in chapter NamingConvention. With this identifier the context is setup matching the requested algorithm. Here, the setup can influence the organization of the cryptographic material, the provided internal buffers for keys, input, or output data and the buffers length. Some cryptographic algorithms need specific initialization parameters. All the specific needs of an algorithm are specified by the corresponding standards, and provide details on how to internally setup the `Crypto Provider` and its supported `Cryptographic primitives`.

### 7.4.1.1 Random Number Generator (RNG)

Generating randomness or pseudo randomness is required for many operations such as creating `Salts` or `Nonces`. In order to enable applications to perform these operations, `CryptoAPI` provides an interface to generate random data.

Randomness can be generated by True `Random Number Generators` (TRNGs) or by Cryptographically Secure Pseudo `Random Number Generators` (CSPRNGs). CSPRNGs hold an internal state that needs to be securely seeded with sufficient entropy. This entropy is used to generate a deterministic but unpredictable stream of random data. More information on the desired properties of CSPRNGs can be found in [8, BSIDRNG: Functionality Classes and Evaluation Methodology for Deterministic `Random Number Generators`].

**[SWS_CRYPT_00501]**{DRAFT} ⌈If a `Crypto Provider` provides one or more random generator implementations, one random generator implementation shall be documented as the default and a corresponding RandomGeneratorCtx shall be returned when `ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx` is called with `algId` == kAlgIdDefault.

If a `Crypto Provider` provides one or more `RNG` implementations, one `RNG` implementation shall be documented as the default. If `ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx` is called with the `algId` parameter equal to kAlgIdDefault , it shall return the default `RNG` implementation context.⌋ *(RS_CRYPTO_02206)*

The definition of the default `RNG` and its implementation is not specified in this document.

Each `ara::crypto::cryp::RandomGeneratorCtx` may either rely on state local to the `ara::crypto::cryp::RandomGeneratorCtx` instance only, or may rely

on global state shared among different `ara::crypto::cryp::RandomGenera-torCtx`'s instances. In order to prevent malicious applications from being able to predict random data generated for other processes, it is important to ensure that applications must not modify the global state of any `ara::crypto::cryp::RandomGeneratorCtx`.

**[SWS_CRYPT_00502]**{DRAFT} ⌈If a `ara::crypto::cryp::RandomGenera-torCtx` uses global state or local state without support for user-provided seed/entropy, calls to its methods `ara::crypto::cryp::RandomGeneratorCtx::Seed`, `ara::crypto::cryp::RandomGeneratorCtx::SetKey` and `ara::crypto::-cryp::RandomGeneratorCtx::AddEntropy` shall return false without modifying the global state or local state.⌋*(RS_CRYPTO_02206)*

**[SWS_CRYPT_00503]**{DRAFT} ⌈`ara::crypto::cryp::RandomGenera-torCtx::Seed`, and `ara::crypto::cryp::RandomGeneratorCtx::SetKey` shall return false without modifying the state, if they are called with a `Symmetric Key` or a `ara::crypto::cryp::SecretSeed` without the allowed usage flag `kAllowRngInit`.⌋*(RS_CRYPTO_02206)*

How global-state `ara::crypto::cryp::RandomGeneratorCtx`s are seeded is stack-vendor and/or project specific and out of scope of this specification. Local-state `ara::crypto::cryp::RandomGeneratorCtx`'s may be seeded by `FC Crypto`.

**[SWS_CRYPT_00504]**{DRAFT} ⌈If `ara::crypto::cryp::CryptoProvider::-CreateRandomGeneratorCtx` is called to create a local-state `ara::crypto::-cryp::RandomGeneratorCtx` with `initialize` set to true, the internal state of the created `ara::crypto::cryp::RandomGeneratorCtx` shall be seeded by `FC Crypto` before returning.⌋*(RS_CRYPTO_02206)*

While this enables applications to create a ready-to-go `ara::crypto::cryp::Ran-domGeneratorCtx`, it cannot be guaranteed that seeding of the `ara::crypto::-cryp::RandomGeneratorCtx` is possible at this point in time, e.g., due to a lack of entropy.

**[SWS_CRYPT_00505]**{DRAFT} ⌈If `ara::crypto::cryp::CryptoProvider::-CreateRandomGeneratorCtx` is called to create a local-state `ara::crypto::-cryp::RandomGeneratorCtx` with `initialize` set to true but the context currently cannot be seeded, `ara::crypto::cryp::CryptoProvider::CreateRandom-GeneratorCtx` shall return `kBusyResource`.⌋*(RS_CRYPTO_02206)*

As applications shall be prevented from modifying the state of global-state `ara::-crypto::cryp::RandomGeneratorCtx`, applications shall also not be able to trigger the seeding of any global-state `ara::crypto::cryp::RandomGeneratorCtx`.

A `ara::crypto::cryp::RandomGeneratorCtx` may have insufficient entropy to serve a request for random data, e.g., because it has not been seeded or because it ran out of entropy. In these cases, `ara::crypto::cryp::RandomGeneratorCtx::-Generate` shall return errors.

**[SWS_CRYPT_00507]**{DRAFT} ⌈If a call to `ara::crypto::cryp::RandomGeneratorCtx::Generate` of a global-state `ara::crypto::cryp::RandomGeneratorCtx` cannot be served with the requested number of random bytes, `kBusyResource` shall be returned.

⌋*(RS_CRYPTO_02206)*

**[SWS_CRYPT_00508]**{DRAFT} ⌈If a call to `ara::crypto::cryp::RandomGeneratorCtx::Generate` of a local-state `ara::crypto::cryp::RandomGeneratorCtx` cannot be served with the requested number of random bytes, `kUninitializedContext` shall be returned.⌋*(RS_CRYPTO_02206)*

These errors represent the possible handling of the error by applications: For a global-state `ara::crypto::cryp::RandomGeneratorCtx` the application has to wait, whereas for a local-state `ara::crypto::cryp::RandomGeneratorCtx` the application has to provide additional entropy.

**[SWS_CRYPT_40983]**{DRAFT} ⌈The function `ara::crypto::GenerateRandomData` shall return random data generated from the default random data source of FC Crypto or `kBusyResource` if the requested number of random bytes cannot be provided.⌋*(RS_CRYPTO_02206)*

#### 7.4.1.2 Key Derivation Function (KDF)

According to [9], [10], [11], and [12] the Key Derivation Function (`KDF`) shall prevent that an attacker, when a derived key was obtained, will gather information about the master secret value or other derived keys. It is also important to strengthen the derived key to prevent an attacker to guess or to brute force the derived key. Therefore, good keys are derived by adding a `Salt`, which avoids dictionary attacks, and a number of iterations, which increase the guessing delay.

**[SWS_CRYPT_00603]**{DRAFT} **Symmetric encryption based `KDF`** ⌈Beside the usage of hashes, the `FC Crypto` shall allow to parametrize symmetric encryption algorithms as the used key derivation function. This is done by the algorithm identifier as well.⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_00608]**{DRAFT} ⌈The interface `ara::crypto::cryp::KeyDerivationFunctionCtx::AddSalt` shall add a `Salt` value stored in the provided non-secret ReadOnlyMemRegion for subsequent key derivation

- `ara::crypto::cryp::KeyDerivationFunctionCtx::AddSalt` shall return a `kInvalidInputSize` error, if the size of the provided `Salt` is not supported by the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02101)*

The `CryptoAPI` provides the `ara::crypto::cryp::KeyDerivationFunctionCtx::AddSalt` interface in the `KDF` context. Deriving the key is done by the given target symmetric algorithm identifier, which also defines a length of derived key.

**[SWS_CRYPT_00609]**{DRAFT}  ⌈The interface `ara::crypto::cryp::KeyDerivationFunctionCtx::AddSecretSalt` shall add a secret `Salt` value stored in the provided `ara::crypto::cryp::SecretSeed` for subsequent key derivation

- `ara::crypto::cryp::KeyDerivationFunctionCtx::AddSecretSalt` shall return a `kInvalidInputSize` error, if the size of the provided secret `Salt` is not supported by the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_00610]**{DRAFT}  ⌈The interface `ara::crypto::cryp::KeyDerivationFunctionCtx::ConfigIterations` shall configure the number of iterations for subsequent key derivation. If the provided number of iterations is smaller or larger than the implementation of this interface supports, the interface shall return the actual number of iterations applied otherwise the interface shall return the provided number of iterations.⌋*(RS_CRYPTO_02101)*

The stack vendor may restrict the maximum number of iterations to avoid overloading the system. The stack vendor may enforce a minimum number of itertions needed to derive a secure key.

**[SWS_CRYPT_00611]**{DRAFT}  ⌈The interfaces `ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveKey` and `ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveSeed` shall apply the configured key derivation algorithm for the provided context configuration. The interface shall return the derived `Key Material` as a `ara::crypto::cryp::SymmetricKey` or `ara::crypto::cryp::SecretSeed` respectively. The returned objects shall be configured according to the provided flags `isSession` or `isExportable` and `isSession` or `isExportable`. If the flags are not provided the object instance shall be session and not exportable. The interface shall return `kUninitializedContext`, if the configured key derivation algorithm requires more context configuration than provided.⌋ *(RS_CRYPTO_02101)*

**[SWS_CRYPT_00622]**{DRAFT} **Signalization of error** ⌈By conventions, if any algorithm fails the `FC Crypto` shall provide a distinct error. The context will fail:

- If the input or output lengths exceed some (very large) implementation defined bound.

⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_40944]**{DRAFT}  ⌈The interfaces `ara::crypto::cryp::KeyDerivationFunctionCtx::SetSourceKeyMaterial` and `ara::crypto::cryp::KeyDerivationFunctionCtx::SetSourceKeyMaterial` shall deploy the provided data (`ara::crypto::cryp::RestrictedUseObject` or `ara::crypto::cryp::ReadOnlyMemRegion`) as source input for key derivation. The interface shall return

- `kUsageViolation` error, if the allowed usage flag `kAllowKdfMaterial` of the provided `ara::crypto::cryp::RestrictedUseObject` is not set.

- `kIncompatibleObject` error, if the provided `ara::crypto::cryp::RestrictedUseObject` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kBruteForceRisk` error, if the provided source material is below a implementation defined size

⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_40945]**{DRAFT} ⌈The interface `ara::crypto::cryp::KeyDerivationFunctionCtx::Init` shall configure the key derivation by setting the provided `targetKeyId`, the optionally provided `ara::crypto::CryptoAlgId`, usage flags and context label of the derived key. The interface shall return:

- `kUsageViolation` error, if a `ara::crypto::cryp::RestrictedUseObject` has been provided as source `Key Material` and its allowed usage flags are more restrictive than the allowed usage flags provided by this interface.

- `kIncompatibleArguments` error, if the provided `targetAlgId` is incompatible with the size of the `Key Material` derived by the configured key derivation algorithm.

⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_40946]**{DRAFT} ⌈The interface `ara::crypto::cryp::KeyDerivationFunctionCtx::GetTargetAllowedUsage` shall return the allowed usage flags of the derived key.

- If the context has not yet been configured by a call to `ara::crypto::cryp::KeyDerivationFunctionCtx::Init` and a `ara::crypto::cryp::RestrictedUseObject` has been provided as source `Key Material`, the allowed usage flags of the source key-material shall be returned.

- If the context has not yet been configured by a call to `ara::crypto::cryp::KeyDerivationFunctionCtx::Init` and no `ara::crypto::cryp::RestrictedUseObject` has been provided as source `Key Material`, `kAllowKdfMaterialAnyUsage` shall be returned.

- If the context has been configured by a call to `ara::crypto::cryp::KeyDerivationFunctionCtx::Init`, the provided `ara::crypto::AllowedUsageFlags` shall be returned or `kAllowKdfMaterialAnyUsage` in case `ara::crypto::AllowedUsageFlags` have not been provided.

⌋*(RS_CRYPTO_02101)*

### 7.4.1.3 Hashing

A hash-function is a one-way function and maps an arbitrary string of bits to a fixed-length string of bits. Due to its nature the bit string result is practical infeasible to invert. Hash-functions are basic elements of cryptography functions. Therefore, the

`FC Crypto` allows application and `Functional Clusters` to use common hash-functions and expose access via the `CryptoAPI` to the user. The `FC Crypto` ensures that the typical properties of modern hash-functions are met and not altered by third parties. The typical properties of modern hash-functions are:

- Determinism: the same input to the hash-function generates always the same result.

- Speed: results are quick to compute.

- No revert: the result is infeasible to revert to the input.

- Collision freedom: two different inputs generate different output.

- Correlation freedom: a small change to the input changes the output significant without providing a correlation of all parts.

**[SWS_CRYPT_00902]**{DRAFT} ⌈The `ara::crypto::cryp::HashFunctionCtx` shall implement hashing.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_00903]**{DRAFT} ⌈The `ara::crypto::cryp::HashFunctionCtx` shall store the calculated hash value until this `ara::crypto::cryp::HashFunctionCtx` object is destroyed or the function `ara::crypto::cryp::HashFunctionCtx::Start` is called again.⌋*(RS_CRYPTO_02205)*

**[SWS_CRYPT_00908]**{DRAFT} **Start** ⌈The functions `ara::crypto::cryp::HashFunctionCtx::Start`, `ara::crypto::cryp::HashFunctionCtx::Start`, `ara::crypto::cryp::HashFunctionCtx::Start` shall clear the current hash value and initialize the context with the provided `IV`.

- `ara::crypto::cryp::HashFunctionCtx::Start`, `ara::crypto::cryp::HashFunctionCtx::Start` shall return a

- `kInvalidInputSize` error, if the size of the provided `IV` is not supported by the configured context `ara::crypto::CryptoAlgId`.

- `ara::crypto::cryp::HashFunctionCtx::Start`, `ara::crypto::cryp::HashFunctionCtx::Start` shall return a `kUnsupported` error, if the configured context `ara::crypto::CryptoAlgId` does not support an `IV`.

- `ara::crypto::cryp::HashFunctionCtx::Start` shall return a `kMissingArgument` error, if the configured context `ara::crypto::CryptoAlgId` expected an `IV` but none was provided.

⌋*(RS_CRYPTO_02205)*

Note, Start method can be called after Update method. In this case the `ara::crypto::cryp::HashFunctionCtx` will not return an error, instead Start method will start a new hash value calculation.

Some `Cryptographic primitives` require an Initialization Vector to guarantee randomness or freshness during the data processing. When an application or `Functional Cluster` specifies a cryptographic primitive, which requires an `IV`, the caller must provide the `IV`.

Hash-function calculation can be resource intensive when the input data has an arbitrary length, which may exceed some (very large) implementation defined bound. A solution is to generate hashes incrementally by presenting parts of the input data, which is hashed. This elementary characteristic is based on two reasons:

- Commonly in practice the entire hash object is not in one contiguous segment available. Instead, often parts are used independently as given by the `HMAC` function for example. Here, the inner hash is some preprocessed keying material, followed by the message being `MAC`'ed. Therefore, a temporary buffer consisting of the `HMAC` inner key ("ipad") and the message can be created. However, this is an overhead.

- The incrementally creation allows to run the hash implementation in memory complexity O(1). The needed memory space for calculation is independent of input size. This is very easy to do with current hash function, such as `SHA-2` and `SHA-3`, where, with a small amount of side memory, the hashing processes the message in pieces.

When an application or `Functional Cluster` uses the hash-function of `FC Crypto`, it expects that the `Crypto Provider` supports this elementary characteristic and the `CryptoAPI` exposes the corresponding interface.

**[SWS_CRYPT_00905]**{DRAFT} **Update** ⌈The functions `ara::crypto::cryp::HashFunctionCtx::Update`, `ara::crypto::cryp::HashFunctionCtx::Update`, `ara::crypto::cryp::HashFunctionCtx::Update` shall implement the configured hash algorithm calculation.⌋*(RS_CRYPTO_02205)*

**[SWS_CRYPT_00909]**{DRAFT} **Update** ⌈The user application shall be able to call `Update` multiple times, each time providing a new chunk of data. `Update` shall update the hash value calculation with each new chunk. `Update` shall return a `CryptoErrorDomain::kProcessingNotStarted` error, if `Start` has not been called before.⌋ *(RS_CRYPTO_02205)*

With the support of the incrementally creation characteristics the `FC Crypto` lost the possibility to know when the input data ends. Therefore, the application or `Functional Cluster` needs the possibility to inform the `Crypto Provider` that all parts of the input was provided and no further input must be processed. The `CryptoAPI` supports this signaling with a corresponding interface.

**[SWS_CRYPT_00906]**{DRAFT} **Finish** ⌈The function `ara::crypto::cryp::HashFunctionCtx::Finish` shall finalize the hash value calculation and return the hash value, i.e. no more data may be provided by `Update`.

- `ara::crypto::cryp::HashFunctionCtx::Finish` shall return a `CryptoErrorDomain::kProcessingNotStarted` error, if `Start` has not been successfully called before.

- `ara::crypto::cryp::HashFunctionCtx::Finish` shall return a `CryptoErrorDomain::kInvalidUsageOrder` error, if `Update` has not been called successfully after the last call to `Start`.

⌋*(RS_CRYPTO_02205)*

**[SWS_CRYPT_00910]**{DRAFT} ⌈If `Finish` is called multiple times for the same hash value calculation, then only the first call shall apply the finalizations step; i.e. all other subsequent calls shall only return the hash value.⌋*(RS_CRYPTO_02205)*

If the signature object is produced by a plain hash-function, then the dependent `COUID` of the signature should be set to `COUID` of context. However, the hash algorithm ID field of the signature shall be set according to the used algorithm ID. If the signature object is produced by a keyed `MAC`/`HMAC`/`AE`/`AEAD` algorithm, then the dependence `COUID` of the signature should be set to `COUID` of used `Symmetric Key`. Instead, the hash algorithm ID field of the signature shall be set to an unknown algorithm ID.

**[SWS_CRYPT_00907]**{DRAFT} **Retrieving the hash value** ⌈The functions `ara::crypto::cryp::HashFunctionCtx::GetDigest`, `ara::crypto::cryp::HashFunctionCtx::GetDigest` shall return the finalized hash value or part of the hash value, if the application requested an offset. The offset specifies the first byte that shall be included in the returned buffer.⌋*(RS_CRYPTO_02205)*

**[SWS_CRYPT_00919]**{DRAFT} **Signalization of missing finalization error** ⌈The functions `ara::crypto::cryp::HashFunctionCtx::GetDigest`, `ara::crypto::cryp::HashFunctionCtx::GetDigest` shall return a `kProcessingNotFinished` error, if `ara::crypto::cryp::HashFunctionCtx::Finish` has not been called for the current hash value calculation.⌋*(RS_CRYPTO_02205)*

### 7.4.1.4 Message Authentication Code (MAC)

According to the ISO-9797 [13] `Message Authentication Code` (`MAC`) algorithms are data integrity mechanisms that compute a short string (the Message Authentication Code or MAC) as a complex function of every bit of the data and of a secret key. Their main security property is unforgeability: someone who does not know the secret key should not be able to predict the MAC on any new data string.

`MAC` algorithms can be used to provide data integrity, as defined in defined in [14] and in [15]. Their purpose is the detection of any unauthorized modification of the data such as deletion, insertion, or transportation of items within data. This includes both malicious and accidental modifications. `MAC` algorithms can also provide data origin authentication. This means that they can provide assurance that a message has been originated by an entity in possession of a specific secret key.

In order to support these mechanism, the `FC Crypto` must provide three basic building blocks:

- A key generation algorithm

- An signing algorithm

- A verifying algorithm

The FC Crypto shall support `Message Authentication Code` generation as described in [14] and in [15].

This identifier is encoded with the common name as defined in chapter 7.5. `MAC` algorithms can be constructed from other `Cryptographic primitives`, like cryptographic hash functions (as in the case of HMAC), which are specified in chapter 7.4.1.3, or from `Block Cipher` algorithms, as defined in chapter 7.4.1.5.1. Both variants are supported by the `FC Crypto`. However, the `Crypto Provider` can either directly access the cryptographic algorithm or use the exposed interfaces provided by the `CryptoAPI`.

The context handles two different use cases, when an application or `Functional Cluster` start processing or generation of the hash-value:

- The context was fresh initialized. No former data was stored in the context, so the `Crypto Provider` can start the calculation on the new data stream (depending from the primitive).

- The context was used previously. Thus, previous stored content will be deleted, the context is rest to a fresh initialization state, and the calculation is started on the new given data stream.

Some `Cryptographic primitives` require an Initialization Vector to guarantee randomness or freshness during the data processing. When an application or `Functional Cluster` specifies a cryptographic primitive, which requires an `IV`, as `MAC` algorithms, the caller must provide the `IV`. Otherwise the `Crypto Provider` will throw an error.

**[SWS_CRYPT_01202]**{DRAFT} ⌈At initialization phase the context allows to specify an optional Initialization Vector (`IV`) or `Nonce` value. If `IV` size is greater than maximally by the algorithm supported length, then an `FC Crypto` uses the leading bytes only.⌋ *(RS_CRYPTO_02203)*

**[SWS_CRYPT_01201]**{DRAFT} ⌈The function `ara::crypto::cryp::MessageAuthnCodeCtx::Start` shall initialize the context for a new data stream processing or generation (depending on the the primitive). The function shall return:

- `kUninitializedContext` error, if the context was not initialized by deploying a key.

- `kInvalidInputSize` error, if the size of provided IV is not supported (i.e. if it is not enough for the initialization).

- `kUnsupported` error, if if the base algorithm (or its current implementation) principally does not support the IV variation, but provided IV value is not empty.

⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_01203]**{DRAFT} **Start** ⌈The function `ara::crypto::cryp::MessageAuthnCodeCtx::Start` shall initialize the context for a new data stream processing or generation (depending on the primitive) with a secret seed. If the size of the secret seed size is greater than maximum supported by the algorithm then an implementation may use the leading bytes only from the sequence. The function shall return:

- `kUninitializedContext` error, if the context was not initialized by deploying a key.

- `kInvalidInputSize` error, if the size of provided secret seed is not supported (i.e. if it is not enough for the initialization).

- `kUnsupported` if the base algorithm (or its current implementation) principally does not support the secret seed variation.

- `kUsageViolation` error, if this transformation type is prohibited by the "allowed usage" restrictions of the provided Secret Seed object.

⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_01204]**{DRAFT} **Update** ⌈The functions `ara::crypto::cryp::MessageAuthnCodeCtx::Update`, `ara::crypto::cryp::MessageAuthnCodeCtx::Update`, `ara::crypto::cryp::MessageAuthnCodeCtx::Update` shall update the digest calculation context by a new part of the message. The functions shall return:

- `kProcessingNotStarted` error, if the digest calculation was not initiated by a call of the `ara::crypto::cryp::MessageAuthnCodeCtx::Start` method.

⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_01207]**{DRAFT} **Finish** ⌈The function `ara::crypto::cryp::MessageAuthnCodeCtx::Finish` shall finalize the hash value calculation, return the hash value and optionally produce the "signature" object. After the call of this function no more data can be provided by calling `ara::crypto::cryp::MessageAuthnCodeCtx::Update`. The function shall return:

- `kProcessingNotStarted` error, if `ara::crypto::cryp::MessageAuthnCodeCtx::Start` has not been successfully called before.

- `kUsageViolation` error, if `ara::crypto::cryp::MessageAuthnCodeCtx::Update`, `ara::crypto::cryp::MessageAuthnCodeCtx::Update`, `ara::crypto::cryp::MessageAuthnCodeCtx::Update` has not been called successfully after the last call to `ara::crypto::cryp::MessageAuthnCodeCtx::Start`.

⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_01208]**{DRAFT} ⌈If the signature object is produced by a plain hash-function then the dependence `COUID` of the "signature" should be set to `COUID` of the used context. But the hash algorithm ID field of the signature should be set according to the used algorithm ID.⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_01209]**{DRAFT} ⌈If the signature object is produced by a keyed `MAC`/ `HMAC`/`AE`/`AEAD` algorithm, then the dependent `COUID` of the signature should be set to `COUID` of the used `Symmetric Key`. However, the hash algorithm ID field of the signature should be set to unknown.⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_01210]**{DRAFT} **GetDigest** ⌈The functions `ara::crypto::cryp:-` `:MessageAuthnCodeCtx::GetDigest`, `ara::crypto::cryp::MessageAuth-` `nCodeCtx::GetDigest` shall provide the hashed output. The `CryptoAPI` allows also to specific an offset. This offset informs the `Crypto Provider` where the position of first byte of digest is that should be placed to the output buffer. The functions shall return:

- `kProcessingNotFinished` error, if the digest calculation was not finished by a call of the Finish() method.

- `kUsageViolation` error, if the buffered digest belongs to a MAC/H-MAC/AE/AEAD context initialized by a key without `kAllowSignature` permission.

⌋*(RS_CRYPTO_02203)*

The `Key Storage Provider` generates and manages the key as described in chapter 7.4.2.2. The key can either be generated or configured in the context of the application or `Functional Cluster`. When the `FC Crypto` provides the context no key is given. The application or `Functional Cluster` will provide the key. The key itself contains also the encoding as an attribute and will not provided by the application or `Functional Cluster` in the call of the `CryptoAPI` method.

**[SWS_CRYPT_01211]**{DRAFT} **SetKey** ⌈The function `ara::crypto::cryp::Mes-` `sageAuthnCodeCtx::SetKey` shall set (deploy) a key to `ara::crypto::cryp:-` `:MessageAuthnCodeCtx`. The function shall return:

- `kIncompatibleObject` error, if the provided key object is incompatible with this `Symmetric Key` context.

- `kUsageViolation` error, if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage"restrictions of provided key object.

⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_01213]**{DRAFT} **Verify** ⌈The `CryptoAPI` shall `ara::crypto::-` `cryp::MessageAuthnCodeCtx::Check` if previous calculated and internally stored `MAC` is valid to an expected "signature" object. Validation is successful, if value and

meta-information of the provided "signature" object is identical to calculated digest and current configuration of the context.⌋*(RS_CRYPTO_02203)*

### 7.4.1.5 Symmetric encryption

Symmetric encryption uses a shared secret (e.g., share key) to encrypt and / or decrypt an information. Without knowing the key, the information cannot be understood by anyone. Symmetric cryptography can be categorized by two algorithm classes:

1. `Block Cipher`: Data with a fixed length is transformed (en/decrypted). The system can only process complete blocks of data held in its internal memory.

2. `Stream Cipher`: Information is encrypted as it streams instead of being retained in the system's memory.

#### 7.4.1.5.1 Block cipher

The encryption method, `Block Cipher`, applies an algorithm with a `Symmetric Key` to encrypt an input data. `Block Ciphers` are commonly used to protect data at rest, such as on file systems.

**[SWS_CRYPT_01502]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey` shall configure this context for encryption or decryption according to the provided `ara::crypto::CryptoTransform` and ensure that the provided `ara::crypto::cryp::SymmetricKey` is used for the following en/decryption.

- `SetKey` shall return a `CryptoErrorDomain::kIncompatibleObject` error, if the provided `Symmetric Key` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `SetKey` shall return a `CryptoErrorDomain::kUsageViolation` error, if the provided transformation direction (`CryptoTransform::kEncrypt` or `CryptoTransform::kDecrypt`) does not match the `ara::crypto::AllowedUsageFlags` (`kAllowDataEncryption` or `kAllowDataDecryption`, respectively) of the provided `Symmetric Key`.

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01501]**{DRAFT} ⌈Only the key and transformation direction specified by the last valid call of `ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey` shall be used for the subsequent encryption or decryption operation.⌋*(RS_CRYPTO_02107, RS_CRYPTO_02201)*

**[SWS_CRYPT_01508]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricBlockCipherCtx::GetTransformation` shall return the `ara::crypto::CryptoTransform` that was provided in the last valid call to `ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey`.

- `GetTransformation` shall return a `CryptoErrorDomain::kUninitializedContext` error, if `ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey` was never called.

⌋*(RS_CRYPTO_02107, RS_CRYPTO_02201)*

**[SWS_CRYPT_01506]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricBlockCipherCtx::GetCryptoService` shall return a unique pointer to the `ara::crypto::cryp::CryptoService` associated with this context.⌋*(RS_CRYPTO_02107, RS_CRYPTO_02201)*

**[SWS_CRYPT_01503]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlock` shall apply the configured transformation (encryption or decryption) to the provided `ara::crypto::ReadOnlyMemRegion` and return the result. Note: `ProcessBlock` shall not apply padding, but instead the size of the input buffer must be equal to the block-size.

- `ProcessBlock` shall return a `CryptoErrorDomain::kUninitializedContext` error, if `ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey` was never called.

- `ProcessBlock` shall return a `CryptoErrorDomain::kInvalidInputSize` error, if the size of the input buffer is not equal to the block-size.

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01504]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlocks` shall apply the configured transformation (encryption or decryption) to the provided `ara::crypto::ReadOnlyMemRegion` and return the result. Note: `ProcessBlocks` shall not apply padding, but instead the size of the input buffer must be a multiple of the block-size.

- `ProcessBlock` shall return a `CryptoErrorDomain::kUninitializedContext` error, if `ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey` was never called.

- `ProcessBlock` shall return a `CryptoErrorDomain::kInvalidInputSize` error, if the size of the input buffer is not a multiple of the block-size.

⌋*(RS_CRYPTO_02201)*

#### 7.4.1.5.2 Stream Cipher

A `Stream Cipher` is used for `Symmetric Key` cryptography, or when the same key is used to encrypt and decrypt data. `Stream Ciphers` encrypt pseudo-random sequences with bits of plain-text in order to generate cipher-text, usually with XOR. `Stream Ciphers` are good for fast implementations with low resource consumption. These two features help the defender implement resistance strategies in devices that may not have the resources for a `Block Cipher` implementation. `Stream Ciphers` can be broadly classified into those that work better in hardware and those that work better in software. `Stream Ciphers` are commonly used to protect data in motion, such as encrypting data on the network.

**[SWS_CRYPT_01651]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::GetBlockService` shall return a unique pointer to the `ara::crypto::cryp::BlockService` associated with this context.⌋*(RS_CRYPTO_02107, RS_CRYPTO_02201)*

**[SWS_CRYPT_01658]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::CountBytesInCache` shall return the number of input data bytes currently held in the context cache.⌋*(RS_CRYPTO_02201)*

Note, that the above requirement applies only to block-wise modes when the user supplied input data that is not a multiple of the block-size. In this case the last data chunk, which cannot be processed because it is less than the block-size, must be cached until the next data processing call adds sufficient data to complete the block-size (and continue processing).

**[SWS_CRYPT_01659]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::SetKey` shall configure this context for encryption or decryption according to the provided `ara::crypto::CryptoTransform` and ensure that the provided `ara::crypto::cryp::SymmetricKey` is used for the following en/decryption.

- `SetKey` shall return a `CryptoErrorDomain::kIncompatibleObject` error, if the provided `Symmetric Key` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `SetKey` shall return a `CryptoErrorDomain::kUsageViolation` error, if the provided transformation direction (`CryptoTransform::kEncrypt` or `CryptoTransform::kDecrypt`) does not match the `ara::crypto::AllowedUsageFlags` (`kAllowDataEncryption` or `kAllowDataDecryption`) of the provided `Symmetric Key`.

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01660]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::GetTransformation` shall return the `ara::crypto::CryptoTransform` that was provided in the last valid call to `ara::crypto::cryp::StreamCipherCtx::SetKey`.

- `GetTransformation` **shall return a** `CryptoErrorDomain::kUninitializedContext` **error, if** `ara::crypto::cryp::StreamCipherCtx::SetKey` **was never called.**

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01661]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::IsBytewiseMode` shall return TRUE, if the algorithm specified during context creation supports updating data byte-wise. It shall return FALSE, if the algorithm can process only data in multiples of the block-size.⌋*(RS_CRYPTO_02201)*

Some operation modes of specific `Stream Ciphers` are seekable, e.g., [16, CTR], [17, Salsa20], or [18, Trivium], and others are not. Seekable means that the user can efficiently seek to any position in the data stream in constant time. If the user needs such functionality and it is unclear if the chosen algorithm provides this kind of functionality, the support of such a mode can be queried.

**[SWS_CRYPT_01662]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::IsSeekableMode` shall return TRUE, if the algorithm specified during context creation supports seek operations.⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01653]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::Seek` shall increment/decrement the position of the next byte to process according to the provided `offset`. If the second and optional boolean parameter equals TRUE, `offset` shall be counted from the start of the stream.

- `Seek` **shall return a** `CryptoErrorDomain::kUnsupported` **error, if this context does not support seeking.**

- `Seek` **shall return a** `CryptoErrorDomain::kProcessingNotStarted` **error, if processing was not started by successfully calling** `Start` **or has already been terminated by successfully calling** `FinishBytes`.

- `Seek` **shall return a** `CryptoErrorDomain::kBelowBoundary` **error, if the absolute seek position is negative.**

- `Seek` **shall return a** `CryptoErrorDomain::kInvalidArgument` **error, if the interface** `ara::crypto::cryp::StreamCipherCtx::IsBytewiseMode` **returns FALSE and the** `offset` **is not aligned on the block boundary.**

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01654]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::Start` shall initialize the context either with an optional `ara::crypto::ReadOnlyMemRegion` or a mandatory `ara::crypto::cryp::SecretSeed`. If the size of initialization data is larger than required by the context, only the leading bytes shall be used.

- `Start` **shall return a** `CryptoErrorDomain::kUninitializedContext` **error, if** `SetKey` **was never called on this context.**

- `Start` shall return a `CryptoErrorDomain::kInvalidInputSize` error, if not enough initialization data has been provided.

- `Start` shall return a `CryptoErrorDomain::kUnsupported` error, if the algorithm selected during context creation does not support initialization but initialization data has been provided nonetheless.

- `Start` shall return a `CryptoErrorDomain::kUsageViolation` error, if the transformation direction provided by a call to `ara::crypto::cryp::StreamCipherCtx::SetKey` (`CryptoTransform::kEncrypt` or `CryptoTransform::kDecrypt`) does not match the `ara::crypto::AllowedUsageFlags` (`kAllowDataEncryption` or `kAllowDataDecryption`) of the provided `ara::crypto::cryp::SecretSeed`.

⌋*(RS_CRYPTO_02201)*

`Start` can be called even if processing has already been started by calling for example `ProcessBlocks`. In this case `Start` will cancel the previous transformation and discard the intermediate result, and re-initialize the context for the new transformation.

Note: `ara::crypto::cryp::StreamCipherCtx::Start` must be called even if the selected algorithm does not support initialization. In this case an empty `ara::crypto::ReadOnlyMemRegion` must be provided.

**[SWS_CRYPT_01655]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::ProcessBlocks` shall apply the configured transformation (encryption or decryption) to the provided data. If the data has been provided as a `ara::crypto::ReadOnlyMemRegion`, `ProcessBlock` shall return the processed data. If the data was provided as a `ara::crypto::ReadWriteMemRegion`, the input data shall be overwritten with the processed data. Note: the size of the input or input and output buffer must be a multiple of the block-size.

- `ProcessBlock` shall return a `CryptoErrorDomain::kInvalidUsageOrder` error, if this interface is called after `ara::crypto::cryp::StreamCipherCtx::ProcessBytes` has been called.

- `ProcessBlock` shall return a `CryptoErrorDomain::kInvalidInputSize` error, if the size of the input buffer is not a multiple of the block-size.

- `ProcessBlock` shall return a `CryptoErrorDomain::kProcessingNotStarted` error, if processing was not started by successfully calling `Start` or has already been terminated by successfully calling `FinishBytes`.

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01656]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::ProcessBytes` shall apply the configured transformation (encryption or decryption) to the provided data and return the result. Note: the size of the input buffer does not need to be a multiple of the block-size. Therefore, if `IsBytewiseMode` equals FALSE, `ProcessBytes` shall keep an internal buffer equal in size to the block-size and only process full blocks of data. If a call to this interface left unprocessed data

in the buffer, the subsequent call's input data shall continue filling the buffer until it can be processed.

- `ProcessBlock` shall return a `CryptoErrorDomain::kProcessingNot-Started` error, if processing was not started by successfully calling `Start` or has already been terminated by successfully calling `FinishBytes`.

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_01657]**{DRAFT} ⌈The interface `ara::crypto::cryp::StreamCipherCtx::FinishBytes` shall apply the configured transformation (encryption or decryption) to the provided data for the last and final time, and return the result. If `IsBytewiseMode` equals FALSE and the provided data is insufficient to end processing with a completely filled internal block-size buffer (cache), then padding shall be applied according to the algorithm selected when creating this context. For decryption the padding shall be removed before returning the processed data.

- `FinishBytes` shall return a `CryptoErrorDomain::kProcessingNot-Started` error, if processing was not started by successfully calling `Start` or has already been terminated by successfully calling `FinishBytes`.

⌋*(RS_CRYPTO_02201)*

Some `Stream Cipher` need an exact multiple of the block length in byte. If the length of the data to be encrypted is not an exact multiple, it must be padded to make it so. Available padding schemes are for example, [19, PKCS5], [20, PKCS5], [21, PKCS7], or [22, ANSI X9.23].

### 7.4.1.6 Authenticated Encryption

Authenticated Encryption (`AE`) or Authenticated Encryption with Associated Data (`AEAD`) provide confidentiality and data authenticity simultaneously. `AEAD` adds the ability to check the integrity and authenticity of some Associated Data (AD), also called "additional authenticated data". Additionally, this mechanism adds an `Message Authentication Code` (`MAC`), as described in chapter 7.4.1.4, to conform that encrypted data is authentic.

Note: the class `ara::crypto::cryp::AuthCipherCtx` provides authenticity and confidentiality only for well known algorithm-protocols that derive both their properties from a single `Symmetric Key` (e.g. ChaCha20-Poly1305, aead/gimli24v1 or AES-GCM). To implement a custom authenticated-encryption protocol (following a pattern of Encrypt-then-Mac, Mac-then-encrypt or Encrypt-and-Mac) the classes `ara::crypto::cryp::StreamCipherCtx` and `ara::crypto::cryp::MessageAuthnCodeCtx` can be used.

**[SWS_CRYPT_01800]**{DRAFT} ⌈The functions `ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData`, `ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData`, `ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData` shall return a `CryptoErrorDomain::kInvalidUsageOrder` error, if `ProcessConfidentialData` has already been called.⌋ *(RS_CRYPTO_02207)*

**[SWS_CRYPT_01801]**{DRAFT} ⌈If associated data is provided by calling `ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData`, `ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData`, `ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData`, the `MAC` calculation must be updated with the associated data.⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01802]**{DRAFT} ⌈Calling `UpdateAssociatedData` is optional for the user. In this case the `MAC` shall be calculated over the confidential data only.⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01803]**{DRAFT} ⌈The function `ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData` shall update the calculation of the `MAC` with the confidential data.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01804]**{DRAFT} ⌈If the transformation direction (`ara::crypto::cryp::AuthCipherCtx::GetTransformation`)is `kEncrypt`, `ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData` shall also encrypt the provided `Plaintext` data and return the `Ciphertext`.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01805]**{DRAFT} ⌈If the transformation direction is `kDecrypt`, `ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData` shall also decrypt the provided `Plaintext` data and return the `Plaintext`, only if the calculated `MAC` matches the provided `expectedTag`. If the calculated `MAC` does not match the provided `expectedTag`, `CryptoErrorDomain::kAuthTagNotValid` error shall be returned instead.⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01807]**{DRAFT} ⌈The `ara::crypto::cryp::AuthCipherCtx::SetKey` interface of the `AuthCipherCtx` shall check the allowed-usage flags of the key parameter provided.The function shall return

- a `kUsageViolation` error, if `kAllowDataEncryption` is not set and the transformation direction is `CryptoTransform::kEncrypt`.

- a `kUsageViolation` error, if `kAllowDataEncryption` is not set and the transformation direction is `CryptoTransform::kDecrypt`.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01808]**{DRAFT} ⌈The function `ara::crypto::cryp::AuthCipherCtx::Start` shall initialize the transformation using the provided `IV` or `Nonce`. The function shall return:

- a `kUninitializedContext` error, if `ara::crypto::cryp::AuthCipherCtx::SetKey` has not been called before.

- a `kInvalidInputSize` error, if the provided data is insufficient.

- a `kUnsupported` error, if the `ara::crypto::CryptoAlgId` specified does not support an `IV` or a `Nonce`.

- a `kUsageViolation` error, if a `ara::crypto::cryp::SecretSeed` instance has been provided as the `IV` or `Nonce` and its allowed usage flags (`kAllowDataEncryption` or `kAllowDataDecryption`) do not match the transformation direction set by the `ara::crypto::cryp::AuthCipherCtx::SetKey` function `kEncrypt` or `kDecrypt`.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01811]**{DRAFT} ⌈The `ara::crypto::cryp::AuthCipherCtx::-GetDigest` function shall return the calculated `MAC` as raw data only after the `ProcessConfidentialData` has been successfully executed.⌋*(RS_CRYPTO_02207)*


### 7.4.1.7 Key Wrapping

Key Wrapping (as defined in [23] and [24]) encapsulates `Key Material`, which is used for example to store a key in an unsecure environment or transport a key by an unsecure channel. Wrapping a key is a kind of encryption of the key and contributes to confidentiality.

Wrapping a key requires a `KEK`. With the call of the `CryptoAPI` interface the `KEK` is set (deployed) to the key wrapper algorithm context. Additionally, a "direction" indicator is used to define the transformation direction, such as wrapping, unwrapping, signature calculation, or signature verification.

**[SWS_CRYPT_02121]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricKeyWrapperCtx::CalculateWrappedKeySize` shall calculate the size of the wrapped key based on the provided `keyLength` of the key to wrap and return the result.⌋*(RS_CRYPTO_02208)*

**[SWS_CRYPT_02122]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricKeyWrapperCtx::SetKey` shall configure this context for encryption or decryption according to the provided `ara::crypto::CryptoTransform` and ensure the provided `ara::crypto::cryp::SymmetricKey` is used as the key-encryption-key (KEK) for subsequent processing in this context.

- `SetKey` shall return a `CryptoErrorDomain::kIncompatibleObject` error, if the provided `Symmetric Key` belongs to a different `ara::crypto::cryp:-:CryptoProvider` instance.

- `SetKey` **shall return a** `CryptoErrorDomain::kUsageViolation` **error, if the provided transformation direction** (`CryptoTransform::kEncrypt` **or** `CryptoTransform::kDecrypt`) **does not match the** `ara::crypto::AllowedUsageFlags` (`kAllowKeyExporting` **or** `kAllowKeyImporting`) **of the provided** `SymmetricKey`.

⌋*(RS_CRYPTO_02208)*

**[SWS_CRYPT_02123]**{DRAFT} ⌈Only the key and transformation direction specified by the last valid call of `ara::crypto::cryp::SymmetricKeyWrapperCtx::-SetKey` shall be used for the subsequent encryption or decryption operation.⌋*(RS_-CRYPTO_02107, RS_CRYPTO_02201)*

**[SWS_CRYPT_02104]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricKeyWrapperCtx::GetMaxTargetKeyLength` shall return the maximum bit-length of the payload (key-material) that can be protected by the algorithm specified during context creation.⌋*(RS_CRYPTO_02208)*

**[SWS_CRYPT_02106]**{DRAFT} ⌈The interface `ara::crypto::cryp::SymmetricKeyWrapperCtx::GetTargetKeyGranularity` shall return the granularity in Bytes of the payload (key-material) that can be protected by the algorithm specified during context creation.⌋*(RS_CRYPTO_02208)*

The granularity of key-material refers to the minimum key-size that can be protected and implies that the actual key-size has to be a multiple of this value.

**[SWS_CRYPT_02105]**{DRAFT} **wrap** ⌈The interface `ara::crypto::cryp::SymmetricKeyWrapperCtx::WrapKeyMaterial` shall execute the key-wrap operation on the key-material of the provided `ara::crypto::RestrictedUseObject` and return the result.

- `WrapKeyMaterial` **shall return a** `CryptoErrorDomain::kInvalidInputSize` **error, if the length of the provided** `ara::crypto::RestrictedUseObject` **is unsupported by the algorithm specified during context creation.**

- `WrapKeyMaterial` **shall return a** `CryptoErrorDomain::kUninitializedContext` **error, if** `ara::crypto::cryp::SymmetricKeyWrapperCtx::SetKey` **was never called.**

- `WrapKeyMaterial` **shall return a** `CryptoErrorDomain::kUsageViolation` **error, if the** `kAllowExport` **flag is not set in the** `ara::crypto::AllowedUsageFlags` **of the provided** `ara::crypto::RestrictedUseObject`, **or if the** `kAllowKeyExporting` **flag of the** `ara::crypto::AllowedUsageFlags` **is not set for the** `SymmetricKey` **specified in the** `SetKey` **call.**

⌋*(RS_CRYPTO_02208)*

The flags `ara::crypto::AllowedUsageFlags` (`kAllowKeyExporting` **or** `kAllowKeyImporting`) **are set for the provided** Symmetric Key.

Note: this interface was designed to support for example RFC3394 or RFC5649.

**[SWS_CRYPT_02107]**{DRAFT} **unwrap** ⌈The interface `ara::crypto::cryp::-SymmetricKeyWrapperCtx::UnwrapKey` shall execute the key-unwrap operation on the provided `ara::crypto::ReadOnlyMemRegion` and return a unique smart pointer to the instantiated `ara::crypto::RestrictedUseObject`. `UnwrapKey` shall also apply the provided `ara::crypto::AllowedUsageFlags` and `ara::::crypto::CryptoAlgId` to the created `RestrictedUseObject`.⌋*(RS_CRYPTO_-02208)*

**[SWS_CRYPT_02108]**{DRAFT} **unwrap** ⌈The interface `ara::crypto::cryp::-SymmetricKeyWrapperCtx::UnwrapSeed` shall execute the key-unwrap operation on the provided `ara::crypto::ReadOnlyMemRegion` and return a unique smart pointer to the instantiated `ara::crypto::SecretSeed`. `UnwrapSeed` shall also apply the provided `ara::crypto::AllowedUsageFlags` and `ara::-:crypto::CryptoAlgId` to the created `ara::crypto::cryp::SecretSeed`.⌋ *(RS_CRYPTO_02208)*

**[SWS_CRYPT_02109]**{DRAFT} **error handling during unwrap** ⌈The interfaces `ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapSeed` and `ara::-crypto::cryp::SymmetricKeyWrapperCtx::UnwrapKey` shall

- return a `CryptoErrorDomain::kInvalidInputSize` error, if the length of the provided `ara::crypto::ReadOnlyMemRegion` is unsupported by the algorithm specified during context creation.

- return a `CryptoErrorDomain::kUninitializedContext` error, if `ara::-crypto::cryp::SymmetricKeyWrapperCtx::SetKey` was never called.

- return a `CryptoErrorDomain::kUsageViolation` error, if the `kAllowKey-Importing` flag of the `ara::crypto::AllowedUsageFlags` is not set for the `Symmetric Key` specified in the `SetKey` call.

⌋*(RS_CRYPTO_02208)*


### 7.4.1.8 Digital signatures

Digital signature contributes to goal authenticity when information is transferred. Guaranteeing the authenticity of the information asymmetric cryptography is used, where the information is signed by a private key and verified later by using the matching public key. When the verification is successful, the receiver of the information can be sure that the owner of the private key is the sender of the information.

**[SWS_CRYPT_02411]**{DRAFT} ⌈The `ara::crypto::cryp::MsgRecoveryPub-licCtx` shall implement digital signature verification with message recovery according to [25].⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02412]**{DRAFT} ⌈The `ara::crypto::cryp::SigEncodePriva-teCtx` shall implement digital signature generation with message encoding according to [25].⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02413]**{DRAFT} ⌈The `ara::crypto::cryp::SignerPrivateCtx` shall implement digital signature generation.⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02414]**{DRAFT} ⌈The `ara::crypto::cryp::VerifierPublicCtx` shall implement digital signature verification.⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_01820]**{DRAFT} ⌈The interface `ara::crypto::cryp::SignerPrivateCtx::SetKey` shall ensure the provided `ara::crypto::cryp::PrivateKey` is used in the following signature generation. The interface shall return

- `kUsageViolation` error, if the allowed usage flag `kAllowSignature` of the provided `ara::crypto::cryp::PrivateKey` is not set.

- `kIncompatibleObject` error, if the provided `ara::crypto::cryp::PrivateKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject` error, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PrivateKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01821]**{DRAFT} ⌈The interface `ara::crypto::cryp::VerifierPublicCtx::SetKey` shall ensure the provided `ara::crypto::cryp::PublicKey` is used in the following signature verification. The interface shall return

- `kUsageViolation` error, if the allowed usage flag `kAllowVerification` of the provided `ara::crypto::cryp::PublicKey` is not set.

- `kIncompatibleObject` error, if the provided `ara::crypto::cryp::PublicKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject` error, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PublicKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01822]**{DRAFT} ⌈The interface `ara::crypto::cryp::SigEncodePrivateCtx::SetKey` shall ensure the provided `ara::crypto::cryp::PrivateKey` is used in the following signature generation with message encoding. The interface shall return

- `kUsageViolation` error, if the allowed usage flag `kAllowSignature` of the provided `ara::crypto::cryp::PrivateKey` is not set.

- `kIncompatibleObject` error, if the provided `ara::crypto::cryp::PrivateKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject` error, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PrivateKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_01823]**{DRAFT} ⌈The interface `ara::crypto::cryp::MsgRecoveryPublicCtx::SetKey` shall ensure the provided `ara::crypto::cryp::PublicKey` is used in the following signature verification with message decoding. The interface shall return

- `kUsageViolation` error, if the allowed usage flag `kAllowVerification` of the provided `ara::crypto::cryp::PublicKey` is not set.

- `kIncompatibleObject` error, if the provided `ara::crypto::cryp::PublicKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject` error, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PublicKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_02415]**{DRAFT} **Pre-hashed signing** ⌈The interfaces `ara::crypto::cryp::SignerPrivateCtx::SignPreHashed` and `ara::crypto::cryp::SignerPrivateCtx::SignPreHashed` shall implement the signing algorithm configured for this context without hashing. Note: hashing has already been applied by the user. Both interfaces shall return

- `kProcessingNotFinished` , if a `ara::crypto::cryp::HashFunctionCtx` has been supplied and the hash value computation has not been finished.

- `kUninitializedContext` , if `ara::crypto::cryp::SignerPrivateCtx::SetKey` was not called before.

- `kInvalidInputSize` , if the supplied `ara::crypto::ReadOnlyMemRegion` parameter, `hashValue` or `context` is incompatible with the configured signature algorithm.

- `kInvalidArgument` , if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::HashFunctionCtx` or the directly provided `ara::crypto::CryptoAlgId` is incompatible with the configured signature algorithm.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02416]**{DRAFT} **Signing** ⌈The interface `ara::crypto::cryp::SignerPrivateCtx::Sign` shall implement the signing algorithm configured for this context. The interface shall return

- `kUninitializedContext`, if `ara::crypto::cryp::SignerPrivate-Ctx::SetKey` was not called before.

- `kInvalidInputSize`, if a supplied `ara::crypto::ReadOnlyMemRegion` parameter's size is incompatible with the configured signature algorithm.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02417]**{DRAFT} **Pre-hashed verification** ⌈The functions `ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed`, `ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed` and `ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed` shall implement the verification algorithm configured for this context without hashing. Note: hashing has already been applied by the user. All interfaces shall return

- `kProcessingNotFinished`, if a `ara::crypto::cryp::HashFunctionCtx` has been supplied and the hash value computation has not been finished.

- `kUninitializedContext`, if `ara::crypto::cryp::VerifierPublicCtx::SetKey` was not called before.

- `kInvalidInputSize`, if the supplied `ara::crypto::ReadOnlyMemRegion` parameter `hashValue` or `context` or `signature` is incompatible with the configured signature algorithm.

- `kInvalidArgument`, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::HashFunctionCtx` or the directly provided `ara::crypto::CryptoAlgId` is incompatible with the configured signature algorithm.

- `kIncompatibleObject`, if the `ara::crypto::CryptoAlgId` of this context does not match the `ara::crypto::CryptoAlgId` of signature; or the required `ara::crypto::CryptoAlgId` of the hash is not `kAlgIdDefault` and the required hash `ara::crypto::CryptoAlgId` of this context does not match `hashAlgId` or the hash `ara::crypto::CryptoAlgId` of signature.

- `kIncompatibleArguments`, if the provided hash `ara::crypto::CryptoAlgId` is not `kAlgIdDefault` and the `ara::crypto::CryptoAlgId` of the provided signature object does not match the provided hash `ara::crypto::CryptoAlgId`.

- `kBadObjectReference`, if the provided signature object does not reference the public key loaded to the context, i.e. if the COUID of the public key in the context is not equal to the COUID referenced from the signature object.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02418]**{DRAFT} **Truncation of hash value** ⌈The functions `ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed` and `ara::crypto::cryp::SignerPrivateCtx::SignPreHashed` shall truncate the provided hash value, if the bitlength of the provided hash value is larger than the bitlength

used for signing/verification or if the configured algorithm `ara::crypto::CryptoAlgId` used to instantiate this context) allows the use of a hash-value with the provided bitlength and specifies a truncation.⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02419]**{DRAFT} **Signing** ⌈The interface `ara::crypto::cryp::VerifierPublicCtx::Verify` shall implement the verification algorithm configured for this context. The interface shall return

- `kUninitializedContext`, if `ara::crypto::cryp::VerifierPublicCtx::SetKey` was not called before.

- `kInvalidInputSize`, if a supplied `ara::crypto::ReadOnlyMemRegion` parameter's size is incompatible with the configured signature algorithm.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_02421]**{DRAFT} ⌈The interface `ara::crypto::cryp::MsgRecoveryPublicCtx::DecodeAndVerify` shall decode the message from the provided signature and return the message after successful verification according to the configured context `ara::crypto::CryptoAlgId`.⌋*(RS_CRYPTO_02204)*

Note: algorithms that compute a signature over a short message allow to embedd the message inside of the signature. Similarly, the reverse algorithms first decode the message and return it only after successful verification.

**[SWS_CRYPT_02420]**{DRAFT} ⌈The interface `ara::crypto::cryp::SigEncodePrivateCtx::SignAndEncode` shall sign the provided input buffer (message) and encode the message into the generated signature according to the algorithm configured for this context. The interface shall return this signature with encoded message as a vector of bytes or

- `kInvalidInputSize`, if the provided message data is larger than allowed by the configured context `ara::crypto::CryptoAlgId`.

- `kUninitializedContext`, if `ara::crypto::cryp::SigEncodePrivateCtx::SetKey` has not been called before.

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_40984]**{DRAFT} ⌈The interface `ara::crypto::cryp::SigEncodePrivateCtx::GetMaxInputSize` shall return the maximum byte-length of the message that can be signed while also encoding it into the generated signature. If the provided parameter `suppressPadding` equals TRUE, only the number of Bytes available for the message shall be returned. If `suppressPadding` equals FALSE, the returned number shall equal the supported block size.⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_02422]**{DRAFT} ⌈The interface `ara::crypto::cryp::MsgRecoveryPublicCtx::DecodeAndVerify` shall decode the message from the provided signature and return the message only after successful verification according to the algorithm configured for this context. The interface shall return

- `kInvalidInputSize`, if the provided signature data is incomplete. Note: the configured context `ara::crypto::CryptoAlgId` expects more data than provided.

- `kUninitializedContext`, if `ara::crypto::cryp::MsgRecoveryPublicCtx::SetKey` has not been called before.

- `kAuthTagNotValid`, if decoded message could not be verified.

⌋*(RS_CRYPTO_02204)*

The context is generated with an algorithm identifier as specified in chapter 7.5.

### 7.4.1.9  Asymmetric encryption

Asymmetric encryption, asymmetric cryptography, or public key cryptography is a system, which is based on a pair of keys, public key and private key. As the name suggest, a public key can be distributed public to everyone without losing secrecy. Instead, a private key must be kept secret. Compared to symmetric cryptography, every user, who possesses the public key, can encrypt information, but only the user with the private key can decrypt the information.

**[SWS_CRYPT_02700]**{DRAFT} **Separation of asymmetric transformation directions** ⌈The `ara::crypto::cryp::EncryptorPublicCtx` shall implement the asymmetric encryption operation of a `Plaintext` to a `Ciphertext`. The `ara::crypto::cryp::DecryptorPrivateCtx` shall implement the asymmetric decryption operation of a `Ciphertext` to a `Plaintext`. It shall be possible to use both contexts independently.⌋*(RS_CRYPTO_02202)*

The separation of the encryption and decryption context allows an application or `Functional Cluster` to encrypt or decrypt independently based on their needs. When an application or `Functional Cluster` need both, encryption and decryption, it has to setup both contexts.

**[SWS_CRYPT_02701]**{DRAFT} **Creation of DecryptorPrivateCtx and EncryptorPublicCtx** ⌈The interface `ara::crypto::cryp::CryptoProvider::CreateDecryptorPrivateCtx` shall return an instance of `ara::crypto::cryp::DecryptorPrivateCtx` implementing the algorithm specified by the provided parameter `ara::crypto::CryptoAlgId`. The interface shall return

- `kUnknownIdentifier`, if the provided `ara::crypto::CryptoAlgId` is not supported.

- `kInvalidArgument`, if the provided `ara::crypto::CryptoAlgId` is supported but does not refer assymetric decryption hashing.

⌋*(RS_CRYPTO_02202)*

The `ara::crypto::CryptoAlgId` is the implementation specific identifier that represents the algorithm name, as described in chapter 7.5. With this identifier the context

is setup matching the asymmetric algorithm. Here, the setup can influence the organization of the cryptographic material, the provided internal buffers for keys, input, or output data and the buffers length. Some asymmetric cryptographic algorithms need specific initialization parameters. All the specific needs of an asymmetric algorithm, the corresponding standards gives detailed insights how to setup internally the `Crypto Provider` and its supported `Cryptographic primitives`.

The `Key Storage Provider` generates and manages the key as described in chapter 7.4.2.2. The key can either be generated or configured in the context of the application or `Functional Cluster`. When the `FC Crypto` provides the context no key is given. The application or `Functional Cluster` will provide the key. The key itself contains also the encoding as an attribute and will not provided by the application or `Functional Cluster` in the call of the `CryptoAPI` method.

**[SWS_CRYPT_02702]**{DRAFT} ⌈The `ara::crypto::cryp::EncryptorPublicCtx::SetKey` shall check the allowed-usage flags of the key parameter provided. If `kAllowDataEncryption` is not set, a `kUsageViolation` error shall be returned.

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_02703]**{DRAFT} ⌈The `ara::crypto::cryp::DecryptorPrivateCtx::SetKey` shall check the allowed-usage flags of the key parameter provided. If `kAllowDataDecryption` is not set, a `kUsageViolation` error shall be returned.⌋ *(RS_CRYPTO_02202)*

**[SWS_CRYPT_02704]**{DRAFT} **Encrypting** ⌈The interfaces `ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock`, `ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock` shall execute the encryption operation using the deployed public key.⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_02705]**{DRAFT} **Decrypting** ⌈The interface `ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock`, `ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock` shall execute the decryption operation using the deployed public key.⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_02706]**{DRAFT} ⌈If the parameter `suppressPadding` is set to FALSE, the interface `ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock` shall add padding as specified by the `ara::crypto::CryptoAlgId`. If the parameter `suppressPadding` is set to TRUE, the interface `ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock` shall not add any padding.⌋*(RS_CRYPTO_02202)*

If a padding shall be applied or how the padding layout looks like, this is encoded in the common name, as described in chapter 7.5.

**[SWS_CRYPT_02726]**{DRAFT} **Errors of ProcessBlock** ⌈The functions `ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock`, `ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock` shall return

- `kUninitializedContext` error, if `ara::crypto::cryp::EncryptorPublicCtx::SetKey` was not called before.

- `kInvalidInputSize` error, if `suppressPadding` is set to TRUE and the user provided insufficient data.

⌋*(RS_CRYPTO_02202)*

### 7.4.1.10 Key Encapsulation Mechanism (KEM)

Briefly, a key encapsulation mechanism (`KEM`) works just like a public-key encryption scheme, except that the encryption algorithm takes no input other than another key. Therefore, the `KEM` uses randomly generated `Key Material`, the key encryption key (`KEK`), to encapsulate an input, in this situation a key. The input is encapsulated with an encryption with a target public key, as given in [26], [27], and [28]. The `KEK` can be derived from the encapsulated `Key Material` or from randomly generated data by application of a `KDF`.

**[SWS_CRYPT_03000]**{DRAFT}  **Keying-Data** ⌈The interface `ara::crypto::-cryp::KeyEncapsulatorPublicCtx::AddKeyingData` shall set the provided `ara::crypto::cryp::RestrictedUseObject` as payload to be encapsulated (keying-data). The interface shall return

- `kUsageViolation`, if the allowed usage flag `kAllowExport` of the provided `ara::crypto::cryp::RestrictedUseObject` is not set.

- `kIncompatibleObject`, if the provided `ara::crypto::cryp::RestrictedUseObject` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kInvalidInputSize`, if the size of the provided `ara::crypto::cryp::-RestrictedUseObject` is not supported by the configured `ara::crypto:-:CryptoAlgId` of this context.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03002]**{DRAFT}  **Encapsulation** ⌈The interface `ara::crypto:-:cryp::KeyEncapsulatorPublicCtx::Encapsulate` shall execute key-encapsulation according to the configured `ara::crypto::CryptoAlgId` of this context. If the context allows specifying the used key-derivation function and/or the key-encapsulation-key (KEK) primitive, the interface shall override the initial context configuration with the provided `ara::crypto::cryp::KeyDerivationFunctionCtx` and `ara::crypto::CryptoAlgId`. The interface shall return a byte-vector containing the encapsulated keying-data or

- `kUninitializedContext`, if `ara::crypto::cryp::KeyEncapsulatorPublicCtx::SetKey` and `ara::crypto::cryp::KeyEncapsulatorPublicCtx::AddKeyingData` have not been called successfully before.

- `kInvalidArgument`, if the provided `ara::crypto::cryp::KeyDerivationFunctionCtx` or `ara::crypto::CryptoAlgId` are incompatible with the configured `ara::crypto::CryptoAlgId` of this context.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03003]**{DRAFT} **Key Decapsulation** ⌈The interface `ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateKey` shall execute key-decapsulation on the provided `ara::crypto::ReadOnlyMemRegion` according to the configured `ara::crypto::CryptoAlgId` of this context. If the context allows specifying the used key-derivation function and/or the key-encapsulation-key (KEK) primitive, the interface shall override the initial context configuration with the provided `ara::crypto::cryp::KeyDerivationFunctionCtx` and `ara::crypto::CryptoAlgId` (kekAlgId). The interface shall return a non-exportable, non-storable instance of `ara::crypto::cryp::SymmetricKey` representing the decapsulated keying-data with usage restrictions set according to the provided `ara::crypto::AllowedUsageFlags` or `kAllowKdfMaterialAnyUsage`, if `ara::crypto::AllowedUsageFlags` are not provided. The returned object's `ara::crypto::CryptoAlgId` shall be set to the provided `ara::crypto::CryptoAlgId` (keyingDataAlgId). The interface shall return

- `kUninitializedContext`, if `ara::crypto::cryp::KeyEncapsulatorPublicCtx::SetKey` has not been called successfully before.

- `kInvalidArgument`, if the provided `ara::crypto::cryp::KeyDerivationFunctionCtx` or `ara::crypto::CryptoAlgId` are incompatible with the configured `ara::crypto::CryptoAlgId` of this context.

- `kInvalidInputSize`, if the size of the provided `ara::crypto::ReadOnlyMemRegion` is not supported by the configured `ara::crypto::CryptoAlgId` of this context.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03004]**{DRAFT} **Seed Decapsulation** ⌈The interface `ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateSeed` shall execute key-decapsulation on the provided `ara::crypto::ReadOnlyMemRegion` according to the configured `ara::crypto::CryptoAlgId` of this context. The interface shall return a non-exportable, non-storable instance of `ara::crypto::cryp::SecretSeed` representing the decapsulated keying-data with usage restrictions set according to the provided `ara::crypto::AllowedUsageFlags` or `kAllowKdfMaterialAnyUsage`, if `ara::crypto::AllowedUsageFlags` are not provided. The returned object's `ara::crypto::CryptoAlgId` shall be set to the `ara::crypto::CryptoAlgId` of this context. The interface shall return

- `kUninitializedContext`, if `ara::crypto::cryp::KeyEncapsulatorPublicCtx::SetKey` has not been called successfully before.

- `kInvalidInputSize`, if the size of the provided `ara::crypto::ReadOnlyMemRegion` is not supported by the configured `ara::crypto::CryptoAlgId` of this context.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03005]**{DRAFT} ⌈The interface `ara::crypto::cryp::KeyDecapsulatorPrivateCtx::SetKey` shall ensure the provided `ara::crypto::cryp::PrivateKey` is used in the following key decapsulation. The interface shall return

- `kUsageViolation`, if the allowed usage flag `kAllowKeyImporting` of the provided `ara::crypto::cryp::PrivateKey` is not set.

- `kIncompatibleObject`, if the provided `ara::crypto::cryp::PrivateKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject`, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PrivateKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03006]**{DRAFT} ⌈The interface `ara::crypto::cryp::KeyEncapsulatorPublicCtx::SetKey` shall ensure the provided `ara::crypto::cryp::PublicKey` is used in the following key encapsulation. The interface shall return

- `kUsageViolation`, if the allowed usage flag `kAllowKeyExporting` of the provided `ara::crypto::cryp::PublicKey` is not set.

- `kIncompatibleObject`, if the provided `ara::crypto::cryp::PublicKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject`, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PublicKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03007]**{DRAFT} ⌈The interfaces `ara::crypto::cryp::KeyEncapsulatorPublicCtx::GetKekEntropy` and `ara::crypto::cryp::KeyDecapsulatorPrivateCtx::GetKekEntropy` shall return the entropy of the key encapsulation key (KEK) in bits, if a KEK is available or the expected entropy can be computed before KEK generation. The interfaces shall return 0 otherwise.⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03008]**{DRAFT} ⌈The interfaces `ara::crypto::cryp::KeyEncapsulatorPublicCtx::GetEncapsulatedSize` and `ara::crypto::cryp::KeyDecapsulatorPrivateCtx::GetEncapsulatedSize` shall return the size of the encapsulated keying-data in Bytes. The interfaces shall return 0, if the size is unknown at this time, because

- the configured KEM algorithm does not specify a fixed size.

- the keying-data has not been set yet (encapsulation).

- the encapsulated data has not been provided yet (decapsulation).

⌋*(RS_CRYPTO_02209)*

**[SWS_CRYPT_03009]**{DRAFT} ⌈The interfaces `ara::crypto::cryp::KeyEncapsulatorPublicCtx::GetExtensionService` and `ara::crypto::cryp::KeyDecapsulatorPrivateCtx::GetExtensionService` shall return an instance of `ara::crypto::cryp::ExtensionService` that provides information on the configuration of this context at the time the interface was called.⌋*(RS_CRYPTO_-02209)*

#### 7.4.1.11 Key Exchange Protocol, Key Exchange Mechanism, and Key Exchange Scheme

`Key Material` is an essential element of cryptographic algorithms. Therefore, `Key Material` must either be ephemeral (i.e. only temporary) or must be stored persistently in confidential form to ensure it is kept secret. This avoids exposure and misuse. However, there are situations when `Key Material` must be exchanged without actually transmitting the secret (key-material) itself. One example for this is secure communication using symmetric cryptography in the presence of untrusted communication networks and dynamic connections (i.e. communication partners are not known in advance). In such situations the Diffie-Hellman key exchange scheme [29] is the common used key agreement mechanism.

**[SWS_CRYPT_03311]**{DRAFT} **Encryption algorithm** ⌈The `FC Crypto` shall provide an encryption algorithm, which matches the chosen public-private key pair and the key exchange schema.⌋*(RS_CRYPTO_02101)*

**[SWS_CRYPT_03300]**{DRAFT} **Ephemeral key usage** ⌈The interface `ara::crypto::cryp::CryptoProvider::GeneratePrivateKey` shall support the generation of `ara::crypto::cryp::PrivateKey` instances of primitive types matching the `ara::crypto::CryptoAlgId` provided as part of a successful call to `ara::crypto::cryp::CryptoProvider::CreateKeyAgreementPrivateCtx`.⌋*(RS_CRYPTO_02101)*

Note: if a specific algorithm for key agreement is supported by the stack, then also the generation of matching key-material shall be supported to enable ephemeral usage of this scheme.

The `Key Storage Provider` generates and manages the key as described in chapter 7.4.2.2.

**[SWS_CRYPT_03312]**{DRAFT} **SetKey** ⌈The interface `ara::crypto::cryp::KeyAgreementPrivateCtx::SetKey` shall ensure the provided `ara::crypto::cryp::PrivateKey` is used in the following key agreement. The interface shall return

- `kUsageViolation`, if the allowed usage flag `kAllowKeyAgreement` of the provided `ara::crypto::cryp::PrivateKey` is not set.

- `kIncompatibleObject`, if the provided `ara::crypto::cryp::PrivateKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject`, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PrivateKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02102)*

**[SWS_CRYPT_03313]**{DRAFT} ⌈The interface `ara::crypto::cryp::KeyAgreementPrivateCtx::GetExtensionService` shall return an instance of `ara::crypto::cryp::ExtensionService` that provides information on the configuration of this context at the time the interface was called.⌋*(RS_CRYPTO_02103)*

Key agreement requires as input the public key of the communication partner (other side). To retrieve an instance of `ara::crypto::cryp::PublicKey` representing the public key received from the communication partner, the interface `ara::crypto::cryp::CryptoProvider::ImportPublicObject` can be used. Similarly, the communication partner requires the public key of the local application. To send this public data the interface `ara::crypto::Serializable::ExportPublicly` can be used to retrieve the raw data of the public key. Each `ara::crypto::cryp::PublicKey` instance provides this interface.

While the scheme specified here is termed "key agreement", what is actually agreed (or exchanged) is a common shared secret. How this secret data is obtained and used is up to the application. Therefore, the `ara::crypto::cryp::KeyAgreementPrivateCtx` provides two dedicated interfaces to generate a shared secret used for secret seeding or as key-material.

**[SWS_CRYPT_03301]**{DRAFT} **Seed agreement** ⌈The interface `ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeSeed` shall execute the key agreement scheme specified at the creation of this context using the provided `ara::crypto::cryp::PublicKey`. The interface shall return a non-exportable, non-storable instance of `ara::crypto::cryp::SecretSeed` representing the calculated shared secret and restrict the object allowed usage according to the provided allowed usage flags or to `kAllowKdfMaterialAnyUsage`, in case allowed usage flags are not provided. The returned object's `ara::crypto::CryptoAlgId` shall be set to the `ara::crypto::CryptoAlgId` of this context. The interface shall return

- `kUninitializedContext`, if `ara::crypto::cryp::KeyAgreementPrivateCtx::SetKey` was not successfully called before.

- `kIncompatibleObject`, if the provided `ara::crypto::cryp::PublicKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject`, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PublicKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02104)*

**[SWS_CRYPT_03302]**{DRAFT} **Key agreement** ⌈The interface `ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeKey` shall execute the key agreement scheme specified at the creation of this context using the provided `ara::crypto::cryp::PublicKey` and return a non-exportable, non-storable instance of `ara::crypto::cryp::SymmetricKey`. The returned `ara::crypto::cryp::SymmetricKey` shall be restricted according to the provided allowed usage flags as well as to the provided `ara::crypto::CryptoAlgId`. The interface shall return

- `kUninitializedContext`, if `ara::crypto::cryp::KeyAgreementPrivateCtx::SetKey` was not successfully called before.

- `kIncompatibleObject`, if the provided `ara::crypto::cryp::PublicKey` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `kIncompatibleObject`, if the `ara::crypto::CryptoAlgId` of the provided `ara::crypto::cryp::PublicKey` is not compatible with the `ara::crypto::CryptoAlgId` used to instantiate this context.

⌋*(RS_CRYPTO_02105)*

**[SWS_CRYPT_03303]**{DRAFT} **Key agreement - no KDF** ⌈If the transformation specified by the AlgId used to create this context does not include a KDF and no `ara::crypto::cryp::KeyDerivationFunctionCtx` is explicitly provided by calling `ara::crypto::cryp::KeyAgreementPrivateCtx::SetKDF`, the interface `ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeKey` and `ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeSeed` shall return the calculated shared secret. Otherwise the output of key-derivation using the shared secret as input shall be returned.⌋*(RS_CRYPTO_02105)*

**[SWS_CRYPT_03304]**{DRAFT} **Key agreement - optional call parameters** ⌈The interface `ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeKey` shall only process the optionally provided parameters `ara::crypto::ReadOnlyMemRegion Salt` and `ara::crypto::ReadOnlyMemRegion` ctxLabel, if required by the configured `ara::crypto::CryptoAlgId` of this context. If such parameters are required, but not provided, an empty value shall be used.⌋*(RS_CRYPTO_02105)*

**[SWS_CRYPT_03305]**{DRAFT} **Key agreement - KDF** ⌈The interface `ara::crypto::cryp::KeyAgreementPrivateCtx::SetKDF` shall configure this context to derive the final `ara::crypto::cryp::SymmetricKey` or `ara::crypto::cryp::SecretSeed` using the provided `ara::crypto::cryp::KeyDerivationFunctionCtx` from the computed shared secret.⌋*(RS_CRYPTO_02115)*

#### 7.4.1.12 Identification of cryptographic primitives and using one

`Cryptographic primitives` are the basic building blocks of cryptographic systems. These well-established and frequently used elements can be implemented in hardware or software. Every implementation can be independent from each other and provided by different vendors. Implementations are represented by `Crypto`

`Provider`. This kind of decoupling provides some negative impacts. Every vendor can choose the `Cryptographic primitives` and their names independently. Then, during development phase of application or `Functional Cluster`, it is not clear how to access the needed algorithm. Therefore, a common name is specified, which allows to develop functionality independent from `FC Crypto`. The common name of the algorithm is given in chapter 7.5. With this common name, it is possible to bind the application or function cluster to the `FC Crypto` during integration phase. However, this approaches needs both, the interface to translate the common name to a vendor specific name and the support from the `FC Crypto`.

**[SWS_CRYPT_03904]**{DRAFT} ⌈The `ara::crypto::cryp::CryptoContext::-GetCryptoPrimitiveId` shall return a `ara::crypto::cryp::CryptoPrimitiveId` of the current used cryptographic algorithm.⌋*(RS_CRYPTO_02308)*

**[SWS_CRYPT_03905]**{DRAFT} ⌈The `ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveName` shall return the common name of the current used cryptographic algorithm.⌋*(RS_CRYPTO_02308)*

**[SWS_CRYPT_03906]**{DRAFT} ⌈The `ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveId` shall return the `ara::crypto::cryp::CryptoPrimitiveId` of the current used cryptographic algorithm.⌋*(RS_CRYPTO_02308)*

This allows a decoupling of the vendor specific implementation and the using application. With this freedom a late binding during integration phase is realized.

**[SWS_CRYPT_40985]**{DRAFT} ⌈The interface `ara::crypto::cryp::CryptoService::GetMaxInputSize` shall return the maximum byte-length of data that the cryptographic context associated with this `ara::crypto::cryp::CryptoService` expects as input for applying its cryptographic transformation. If the provided parameter `suppressPadding` equals TRUE, only the number of Bytes available for the payload shall be returned. If `suppressPadding` equals FALSE, the returned number shall equal the supported block size.⌋*(RS_CRYPTO_02309)*

Note, several encryption algorithms require a certain number of bytes to be added as padding to improve cryptographic properties (e.g. RSA encryption PKCS1 v1.5)

### 7.4.1.13  Support on internal elements (Loading, Update, Import, and Export)

**[SWS_CRYPT_04200]**{DRAFT}  **Loading cryptographic material** ⌈The load interfaces `ara::crypto::cryp::CryptoProvider::LoadObject`, `ara::crypto::cryp::CryptoProvider::LoadSymmetricKey`, `ara::crypto::cryp::CryptoProvider::LoadPublicKey`, `ara::crypto::cryp::CryptoProvider::LoadPrivateKey`, `ara::crypto::cryp::CryptoProvider::LoadSecretSeed` shall load the content from the location pointed to by the provided IOInterface and return an instance of type CryptoObject, `Symmetric Key`, PublicKey, PrivateKey and SecretSeed respectively. The load interface shall return

- `kEmptyContainer`, if the underlying resource this IOInterface points to is empty.

- `kResourceFault` , if the underlying resource this IOInterface points to is faulty.

- `kModifiedResource` , if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.

- `kIncompatibleObject` , if the underlying resource belongs to another incompatible CryptoProvider or if the type of the crypto object to be returned by the respective interface does not match the type contained in the underlying resource.

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112, RS_CRYPTO_02113)*

**[SWS_CRYPT_40947]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-GetAllowedUsage` shall return the allowed usage flags of the underlying CryptoObject this IOInterface points to. If the content that the IOInterface points to is empty, `kAllowPrototypedOnly` shall be returned.⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40948]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-GetCapacity` shall return capacity of the underlying resource in bytes.⌋*(RS_-CRYPTO_02004)*

Note: IOInterfaces always point to an underlying resource to store CryptoObjects such as the RAM buffer of a VolatileTrustedContainer or the persistent memory of a KeySlot. In both cases the underlying resource has a maximum capacity to store a CryptoObject and the content may be empty.

**[SWS_CRYPT_40949]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-GetCryptoObjectType` shall return the CryptoObjectType of the underlying CryptoObject this IOInterface points to. In case the underlying resource this IOInterface points to is empty, `kUndefined` shall be returned.⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40950]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-GetPayloadSize` shall return size of the underlying CryptoObject's key-material this IOInterface points to in bytes. The interface shall return 0, if the container is empty.⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40951]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-GetPrimitiveId` shall return the vendor specific `ara::crypto::CryptoAlgId` of the underlying CryptoObject this IOInterface points to. If the underlying resource this IOInterface points to is empty, `kAlgIdUndefined` shall be returned.⌋*(RS_CRYPTO_-02004)*

**[SWS_CRYPT_40952]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-GetTypeRestriction` shall return the CryptoObjectType that is allwed to be stored in the underlying resource this IOInterface points to. The interface shall return `kUndefined`

- if this IOInterface points to a VolatileTrustedContainer.

- if this IOInterface points to a `Key Slot` and the KeySlot's `mAllowContentTypeChange` flag is set to TRUE.

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40953]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-IsObjectExportable` shall only return TRUE, if `kAllowExport` is set in the allowed usage flags of the CryptoObject stored in the underlying resource this IOInterface points to.⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40954]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-IsObjectSession` shall return TRUE, if the CryptoObject stored in the underlying resource this IOInterface points to is volatile and cannot be persisted (session flag set). The interface shall return FALSE, if the underlying resource this IOInterface points to

- is a `Key Slot`.

- is is empty.

- is volatile but can be persisted.

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40955]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-IsValid` shall only return TRUE, if the underlying resource this IOInterface points to is a VolatileTrustedContainer or a KeySlot that has not been modified since this IOInterface has been obtained by calling `ara::crypto::keys::KeySlot::Open` on the loaded KeySlot instance.⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40956]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-IsVolatile` shall only return TRUE, if this IOInterface points to a VolatileTrustedContainer.⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_40957]**{DRAFT} ⌈The interface `ara::crypto::IOInterface::-IsWritable` shall only return TRUE, if this IOInterface points to a VolatileTrustedContainer or this IOInterface has been obtained by calling `ara::crypto::keys::-KeySlot::Open` with the writable flag set to TRUE.⌋*(RS_CRYPTO_02004)*

The serialization format for exporting/importing is not yet standardized in AUTOSAR.

Therefore it is the responsibility of the platform vendor to adequately de-/serialize CryptoObjects including all relevant meta-data such that CryptoObjects can be transferred between adaptive machines (of the same vendor) without loss of information and functionality.

**[SWS_CRYPT_04202]**{DRAFT} **Exporting secure objects** ⌈The function `ara::-crypto::cryp::CryptoProvider::ExportSecuredObject` shall serialize the provided CryptoObject and apply the transformation specified by the provided SymmetricKeyWrapperCtx. The function shall return the serialized data as a vector of bytes or

- `kIncompatibleObject` if the object cannot be exported due to `ara::-crypto::cryp::CryptoObject::IsExportable` returning FALSE.

- `kIncompleteArgState` if the provided SymmetricKeyWrapperCtx is not fully initialized.

- `kIncompatibleObject` if the flag kAllowKeyExporting of the `Symmetric Key` set in the provided SymmetricKeyWrapperCtx is not set to TRUE

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_04213]**{DRAFT} ⌈The function `ara::crypto::cryp::Crypto-Provider::ExportSecuredObject` shall serialize the CryptoObject contained in the storage location pointed to by the provided IOInterface after applying the transformation specified by the provided SymmetricKeyWrapperCtx. The function shall return the serialized data as a vector of bytes or

- `kEmptyContainer` if the underlying resource this IOInterface points to is empty

- `kIncompleteArgState` if the provided SymmetricKeyWrapperCtx is not fully initialized

- `kIncompatibleObject` if the flag kAllowKeyExporting of the `Symmetric Key` set in the provided SymmetricKeyWrapperCtx is not set

- `kModifiedResource` if this IOInterface points to an instance of a KeySlot that has been modified after the IOInterface has been opened.

⌋*(RS_CRYPTO_02006)*

**[SWS_CRYPT_04203]**{DRAFT} **Exporting public objects** ⌈The function `ara::crypto::cryp::CryptoProvider::ExportPublicObject` shall serialize the CryptoObject contained in the storage location pointed to by the provided IOInterface. The function shall return the serialized data as a vector of bytes or

- `kEmptyContainer` if the underlying resource this IOInterface points to is empty.

- `kUnexpectedValue` if the underlying resource this IOInterface points contains a RestrictedUseObject.

- `kModifiedResource` if this IOInterface points to an instance of a KeySlot that has been modified after the IOInterface has been opened.

⌋*(RS_CRYPTO_02004)*

Both ExportSecuredObject interfaces can export internal objects in a secure manner. This allows exchanging cryptographic objects between platforms or different applications without exposing them to third parties.

**[SWS_CRYPT_40958]**{DRAFT} ⌈The function `ara::crypto::cryp::CryptoObject::IsExportable` shall only return TRUE, if `kAllowExport` is set in the allowed usage flags of this CryptoObject.⌋*(RS_CRYPTO_02006)*

**[SWS_CRYPT_04204]**{DRAFT} **Importing secure objects** ⌈The function `ara::crypto::cryp::CryptoProvider::ImportSecuredObject` shall unwrap securely serialized data provided by the application according to the specified SymmetricKeyWrapperCtx. The unwrapped CryptoObject shall be deserialized and saved to the persistent or volatile storage represented by the provided IOInterface. The function shall return

- `kUnexpectedValue` if the payload (serialized CryptoObject) contains invalid data (errors in the cipher-text or plain-text), or the unwrapping operation failed.

- `kBadObjectType` if the contained CryptoObject does not match the provided CryptoObjectType.

- `kIncompleteArgState` if the provided SymmetricKeyWrapperCtx is not fully initialized.

- `kUsageViolation` if the flag kAllowKeyImporting of the `Symmetric Key` set in the provided SymmetricKeyWrapperCtx is not set to TRUE.

- `kInsufficientCapacity` if the capacity of the underlying resource pointed to by the provided IOInterface is insufficient to hold the deserialized CryptoObject.

- `kUnreservedResource` if the IOInterface is not opened writable.

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_04205]**{DRAFT} **Importing public objects** ⌈The function `ara::crypto::cryp::CryptoProvider::ImportPublicObject` shall deserialize the provided serialized data and save the contained CryptoObject to the persistent or volatile storage represented by the provided IOInterface. The function shall return

- `kUnexpectedValue` if the payload (serialized CryptoObject and associated meta-data) contains invalid data.

- `kBadObjectType` if the contained CryptoObject does not match the provided exptected CryptoObjectType.

- `kInsufficientCapacity` if the capacity of the underlying resource pointed to by the provided IOInterface is insufficient to hold the deserialized CryptoObject.

- `kUnreservedResource` if the IOInterface is not opened writable.

⌋*(RS_CRYPTO_02004)*

Vulnerability notice: using the interface `ara::crypto::cryp::CryptoProvider::ImportPublicObject` to import secret key-material without confidentiality protection is strongly discouraged.

This is an obvious attack path and may compromise security of the whole platform. It is assumed that all parties involved in such a setup are aware of the risk and implement sufficient countermeasures.

**[SWS_CRYPT_04207]**{DRAFT} ⌈The function `ara::crypto::cryp::Crypto-Provider::GetPayloadStorageSize` shall return the minimum required capacity of a KeySlot for storing a CryptoObject defined by the provided `ara::crypto::-CryptoAlgId` and CryptoObjectType. The function shall return

- `kUnknownIdentifier` if the provided `ara::crypto::CryptoAlgId` is unsupported or the provided `ara::crypto::CryptoAlgId` equals `kUndefined`.

- `kIncompatibleArguments` if the provided pair of `ara::crypto::CryptoAlgId` and CryptoObjectType represents an unsupported combination.

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_04208]**{DRAFT} ⌈The function `ara::crypto::cryp::Crypto-Provider::AllocVolatileContainer` shall allocate a volatile buffer with sufficient size to hold cryptographic data of the provided capacity and the meta-data associated with each CryptoObject. The function shall return an instance of VolatileTrustedContainer representing the allocated buffer or `kInsufficientResource`, if not enough volatile memory is available for allocation.⌋*(RS_CRYPTO_02004)*

This type of containers could be used for execution of import operations described above.

**[SWS_CRYPT_40959]**{DRAFT} ⌈The function `ara::crypto::cryp::Crypto-Provider::AllocVolatileContainer` shall allocate a volatile buffer with sufficient size to hold cryptographic data and the meta-data associated with each CryptoObject. The necessary size of cryptographic data shall be computed from the provided pair of `ara::crypto::CryptoAlgId` and CryptoObjectType. The function shall return an instance of VolatileTrustedContainer representing the allocated buffer or

- `kInsufficientResource`, if not enough volatile memory is available for allocation

- `kInvalidArgument` if the provided pair of `ara::crypto::CryptoAlgId` and CryptoObjectType represents an unsupported combination.

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_04209]**{DRAFT} ⌈The `CryptoAPI` shall document all importing or exporting by a logging mechanism. This information can be queried.⌋*(RS_CRYPTO_-02004)*

### 7.4.2 Key Storage Provider

The Key Storage Provider (`KSP`, namespace ara::crypto::keys) is responsible for secure (confidential and or authentic) storage of different type `Key Material` (public, private, secret keys, or seeds) and other security critical cryptographic objects (digital

signatures, hash, `MAC` HMAC tags). These cryptographic objects are represented as a `Key Slots`.

`Key Slots` used by application are defined by the integrator in the manifest via `CryptoKeySlot`.

`CryptoKeySlotInterface` and `CryptoKeySlotToPortPrototypeMapping`

**[SWS_CRYPT_10000]**{DRAFT} ⌈The `FC Crypto` shall provide access to the `CryptoKeySlot`s for every `AdaptiveApplicationSwComponentType`. Every `CryptoKeySlot` is represented by `RPortPrototype` typed by `CryptoKeySlotInterface` in application design.⌋(*RS_CRYPTO_02004, RS_CRYPTO_02305*)

Assignment of `CryptoKeySlot`s to a `CryptoProvider` is described in the manifest. So with the usage of a `RPortPrototype` that is typed by a `CryptoKeySlotInterface` the assignment to `CryptoProvider` is established.

**[SWS_CRYPT_10003]**{DRAFT} ⌈The `CryptoAPI` shall provide a function to obtain `Crypto Provider`. With a call of `ara::crypto::cryp::CryptoContext::MyProvider` the `FC Crypto` provides the correspinding `Crypto Provider` of a `Key Slot`.⌋(*RS_CRYPTO_02009*)

The manifest contains separate deployment data for each `Process`. The class `CryptoKeySlotToPortPrototypeMapping` defines the mapping between a `Process`, a `CryptoKeySlot`, and an `RPortPrototype`. Furthermore, the class `CryptoProviderToPortPrototypeMapping` defines the mapping between a `Process`, a `CryptoProvider`, and an `RPortPrototype`. Figure 7.7 shows the relevant model elements. Additional model elements and links are only shown for context.
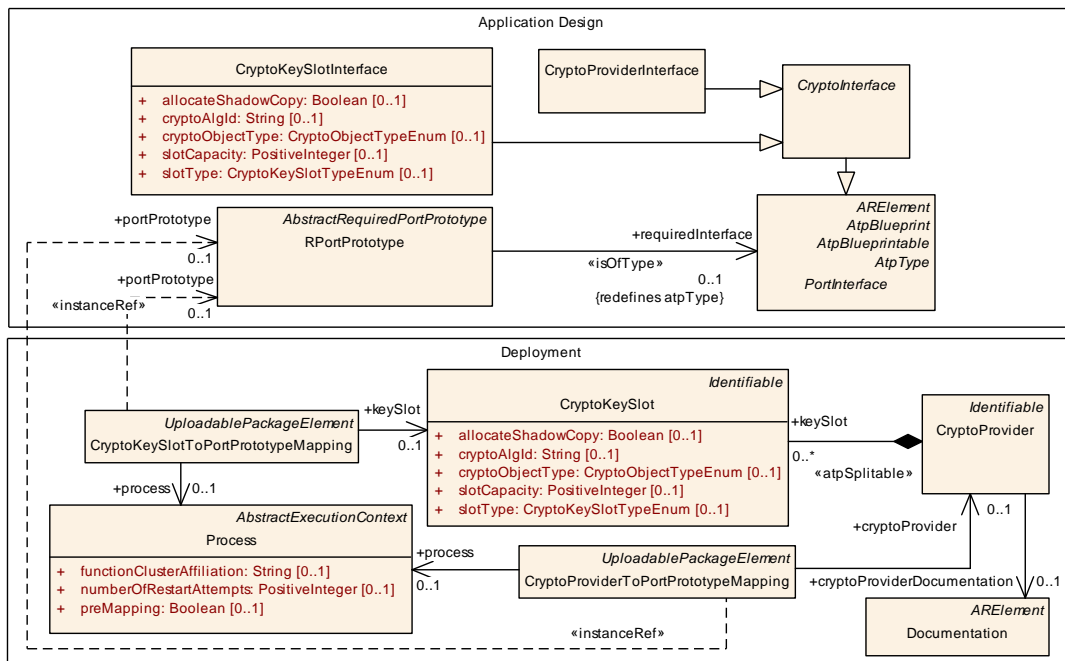


**Figure 7.7: Key deployment**

**[SWS_CRYPT_10005]**{DRAFT} ⌈The `Key Slot` shall be identified during runtime. The `CryptoAPI` provides an interface with a call to `ara::crypto::keys::-KeyStorageProvider::LoadKeySlot` to support this. The interface needs an `ara::core::InstanceSpecifier` as an input parameter. Here, `ara::core:-:InstanceSpecifier` represents a path to `RPortPrototype` mapped to needed `CryptoKeySlot`.⌋*(RS_CRYPTO_02405)*

`CryptoAPI` consumers work with logically single `KSP` that is used for access to all cryptographic objects independently from their physical hosting on the `ECU`. However, from the stack supplier point of view, each `HSM` may support own back-end `KSP` responsible for access control to internally stored cryptographic objects. All back-end `KSP` are hidden from the consumers (under public `CryptoAPI`).

**[SWS_CRYPT_10004]**{DRAFT} ⌈The `FC Crypto` shall ensure confidentiality and authenticity of processed and stored objects with a correct `KSP` implementation (similar to Classic Platform). Thus, its implementation shall be isolated from the consumers' code space.⌋*(RS_CRYPTO_02008, RS_CRYPTO_02009, RS_CRYPTO_02106)*

The "Key Management" functionality is split into four parts:

1. Key Storage Provider API (namespace crypto::keys).

2. Certificate Management Provider API completely (namespace crypto::x509).

3. Key Material Generation, Secured Export, Public/Secured Import and auxiliary API (via methods of crypto::cryp::CryptoProvider interface). These methods represent all actions that need implementation of cryptographic transformations of keys. The usage of `HSM` is implemented in hardware and thus may not support all APIs as software solutions would.

4. Generic serialization of public cryptographic objects (via crypto::Serializable interface). Taking into account the deep dependence of 3rd category of the "Key Management" sub-API from other cryptographic functionality, possibility to reuse some functional blocks (including mechanisms of access control to `Key Material` in `HSM` realms), there is no practical sense to separate this sub-API from `Crypto Provider API`.

Key Storage & Certificate Management are realized by separated interfaces, because they can be implemented completely independent. This allows to combine both provided by different vendors.

### 7.4.2.1 Serializable interface

**[SWS_CRYPT_10200]**{DRAFT} ⌈The `CryptoAPI` shall provide an interfaces `ara::crypto::Serializable::ExportPublicly`, `ara::crypto::Serializable::ExportPublicly` for exporting of any public (by nature) objects, where additional integrity or confidentiality protection are not needed.⌋*(RS_CRYPTO_02112)*

Interfaces of all public (non-confidential) cryptographic objects and certificates that principally support serialization in plain (non-encrypted and non-authenticated) form are derived from the `ara::crypto::Serializable` interface.

Actually, this interface provides only one serialization method `formatId`.

### 7.4.2.2 Key Generation

Key Generation is the process of generating cryptographic keys. There are two types of Key Generation based on the used cryptographic algorithms:

1. Symmetric Algorithms: A symmetric system consists of a key, which is shared between the different parties.

   **[SWS_CRYPT_10300]**{DRAFT} **Symmetric cryptography** ⌈The `FC Crypto` shall support symmetric cryptography.⌋*(RS_CRYPTO_02101)*.

   **[SWS_CRYPT_10301]**{DRAFT} ⌈The `FC Crypto` shall allocate a new `Symmetric Key` object by a call of function `ara::crypto::cryp::CryptoProvider::GenerateSymmetricKey` and fill it by a new randomly generated value. The function shall return

   - `kUnknownIdentifier` error, if the `ara::crypto::CryptoAlgId` has an unsupported value.

   - `kIncompatibleArguments` error, if `allowedUsage` is incompatible with target algorithm specified by `ara::crypto::CryptoAlgId`.

   ⌋*(RS_CRYPTO_02101)*.

2. Asymmetric Algorithms: Asymmetric systems consist of public and private key, which are generated. The public key is used for encryption, key encapsulation, or signature verification. The private key is used for decryption, key encapsulation, key exchange, or digital signature calculation.

   **[SWS_CRYPT_10303]**{DRAFT} **Asymmetric cryptography** ⌈The `FC Crypto` shall support asymmetric cryptography.⌋*(RS_CRYPTO_02101)*.

   **[SWS_CRYPT_10304]**{DRAFT} ⌈The `FC Crypto` shall support the `Asymmetric Key` generation. The `CryptoAPI` provide such functionality. The private key is generated by calling `ara::crypto::cryp::CryptoProvider::GeneratePrivateKey`.⌋*(RS_CRYPTO_02101)*

   **[SWS_CRYPT_10305]**{DRAFT} ⌈The corresponding public key can be obtained from a private key object by `ara::crypto::cryp::PrivateKey::GetPublicKey`. This function is part of the `CryptoAPI`.⌋*(RS_CRYPTO_02002)*

   **[SWS_CRYPT_10306]**{DRAFT} ⌈As private and public key are tightly coupled which each other, they should have the same `COUID`. A common `COUID` shall be shared for both private and public keys.⌋*(RS_CRYPTO_02005)*

### 7.4.2.3 Exporting and Importing of Key Material

Exporting of `Key Material` is sometimes necessary. This is useful during the setup of communication channels, for example. Importing `Key Material` is also important for a later use. Export and Import facilities of `Crypto Provider` are described in 7.4.1.13.

Another use case to export and import `Key Material` is the confidential delivery of `Symmetric Keys`, e.g., transport keys. This technique is called data encapsulation mechanism and provides a "crypto envelope" or "digital envelope" that protects the secrecy and integrity of data using symmetric-key cryptographic techniques concept. The `FC Crypto` provides two contexts, `ara::crypto::cryp::KeyAgreementPrivateCtx` and `ara::crypto::cryp::KeyEncapsulatorPublicCtx`, which implements the data encapsulation mechanism. Additionally, it is possible to assure non-repudiation by adding a digital signature. This is provided via the `ara::crypto::cryp::HashFunctionCtx` and `ara::crypto::cryp::SignerPrivateCtx`. All contexts contains two building blocks:

- The encryption algorithm

- The decryption algorithm

**[SWS_CRYPT_10403]**{DRAFT} ⌈The `FC Crypto` shall provide private key agreement functionality by a specific context. This context is the `ara::crypto::cryp::KeyAgreementPrivateCtx`. The `CryptoAPI` generates this context via an interface. This interface needs an identifier of the target key-agreement cryptographic algorithm to setup the correct context.⌋*(RS_CRYPTO_02105)*

**[SWS_CRYPT_10401]**{DRAFT} ⌈Key agreement private context shall provide functionality to produce a common secret seed `ara::crypto::cryp::SecretSeed`.

⌋*(RS_CRYPTO_02007)*

**[SWS_CRYPT_10402]**{DRAFT} ⌈Key agreement private context shall provide functionality to produce a common symmetric key.⌋*(RS_CRYPTO_02103)*

### 7.4.3 Certificate handling (X.509 Provider)

`X.509` Certificate Management Provider (`X.509 Provider`) is responsible for `X.509` certificates parsing, verification, authentic storage and local searching by different attributes. In addition, `X.509 Provider` is responsible for storage, management, and processing of Certificate Revocation Lists (`CRLs`) and Delta `CRLs`. The `X.509 Provider` supports the preparation of requests, responses, and parsing according to the Online Certificate Status Protocol (`OCSP`) as defined in [30] and [31].

**[SWS_CRYPT_20000]**{DRAFT} ⌈`FC Crypto` supports only a single instance of the `ara::crypto::x509::X509Provider`. As the `X.509 Provider` is completely

independent from `ara::crypto::cryp::CryptoProvider` and `ara::crypto::keys::KeyStorageProvider` implementation details, it is possible that different vendors provide `X.509 Provider` and `Crypto Provider` / `Key Storage Provider`. Therefore, the standardized `CryptoAPI` guarantees interoperability between these independent building blocks. Applications or `Functional Clusters` can access certificates by `CryptoCertificateInterface`, which is provided by `X.509 Provider`.⌋*(RS_CRYPTO_02307)*

Any `FC Crypto` implementation shall include a single `X.509 Provider`. Responsibility of this provider is the support of Public Key Infrastructure (`PKI`) as defined in [32]. A `PKI` contains a root certificate and one or many certificates. Main feature are:

1. Storages of certificates, certification signing requests (`CSRs`), and certificate revocation lists (`CRLs`).

2. Complete parsing of `X.509` certificates and certificate signing requests (`CSR`).

3. Encoding of all public components of certificate signing requests (e.g. Distinguished Names and `X.509` Extensions).

4. Verification of certificates and certification chains (according to current set of trusted certificates).

5. Trust management of the stored certificates.

6. Search of certificates in local storage based on different parameters.

7. Automatic building of the trust chains according to saved certificates, `CRLs`, and trust configuration.

**[SWS_CRYPT_20001]**{DRAFT} ⌈The `CryptoAPI` provides a secure local access to specific information. The minimal information, which shall be accessable, are the specific system name, the private key, which is associated with the caller, the name of the `CA`, which is used as a trust authority, and the `ara::crypto::x509::X509PublicKeyInfo` (or a fingerprint of the public key where a self-certified version is available elsewhere).⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20002]**{DRAFT} ⌈The `ara::crypto::x509::X509Provider` shall store and provide the root `ara::crypto::x509::Certificate` and all needed `CAs` along the `certification path`, together with the reference to the corresponding public and private keys, which are handled by the `ara::crypto::keys::KeyStorageProvider`. All elements, which are relevant for the `certification path`, shall be stored with local access either hard-coded into the software or in a persistent and tamper-proof manner. The decision how to store the elements is based on:

- Updatability of certificates: When certificates shall be exchangeable or revocable, then these are stored in a volatile but persistent storage. Fixed certificates, which stay forever for example, can be stored hard-coded.

- Use case specific: An application or `Functional Cluster` can have pre-configured certificates, which are stored along side the configuration, e.g. in ARXML.

- Project specific

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20003]**{DRAFT} ⌈The `FC Crypto` shall provide all cryptographic algorithms to generate, validate, and process certificates, which are used in the system. Depending on the certificate the `X.509 Provider` uses the corresponding `Crypto Provider`. However, the `X.509 Provider` can either directly access the cryptographic algorithm or use the exposed interfaces provided by the `CryptoAPI`.⌋ *(RS_CRYPTO_02204, RS_CRYPTO_02306)*

**[SWS_CRYPT_20004]**{DRAFT} ⌈The `X.509 Provider` shall support `ASN.1` parsing. Thus it provides an `ASN.1` parser to read the specific syntax of `X.509` certificates. Typical `X.509` certificates must follow the definition given in [32] and [33, RFC 5280]:

1. Certificate

   (a) Version Number

   (b) `ara::crypto::x509::Certificate::SerialNumber`

   (c) Signature Algorithm ID

   (d) `ara::crypto::x509::Certificate::IssuerDn`

   (e) Validity period

      i. `ara::crypto::x509::Certificate::StartTime`

      ii. `ara::crypto::x509::Certificate::EndTime`

   (f) `ara::crypto::x509::BasicCertInfo::SubjectDn`

   (g) Subject Public Key Info

      i. Public Key Algorithm

      ii. Subject Public Key

   (h) Issuer Unique Identifier (optional)

   (i) Subject Unique Identifier (optional)

   (j) Extensions (optional)

2. Certificate Signature Algorithm

3. Certificate Signature

Theses certificates are described by `CryptoServiceCertificate` with all elements.⌋*(RS_CRYPTO_02306)*

The `X.509 Provider` parses certificates when an application or `Functional Cluster` uses the `CryptoAPI` interfaces for importing, storing, or verifying of `CSRs` and certificates. This can be problematic when cross-certification or cross-signing is used. Cross-certification allows to trust one entity in another `PKI`. Here, one part of the `PKI` tree signs a part of another `PKI` tree and vice verse. The `X.509 Provider` shall handle this cross-signing in a correct manner, transparent for the application or `Functional Cluster`.

**[SWS_CRYPT_20005]**{DRAFT} **Freedom of interference during update** ⌈It must be possible to regularly update any key pair of certificates, which are part of a `PKI` tree, without affecting any other key pair of related certificates, which can be also part of the same `PKI` tree or part of an independent tree.⌋*(RS_CRYPTO_02112, RS_CRYPTO_-02306)*

**[SWS_CRYPT_20006]**{DRAFT} ⌈The `X.509 Provider` shall generate certificates, so called self-signed certificates, and `CSRs` based on standardized cryptographic algorithms. A specific algorithm can be chosen by the application or the `Functional Cluster` in the generation call. It shall be ensured that the `Crypto Provider` exposes the needed algorithms. During the `CSR` generation a key pair, public and private key, is generated as well. These keys are stored, by the `Key Storage Provider`. Therefore, the `X.509 Provider` shall use either internally or via exposed interfaces the functionality of the `Key Storage Provider` to create, store, and manage the keys.⌋*(RS_CRYPTO_02306)*

`X.509 Provider` supports two variants of long-term storage types:

1. "Persistent" storage is dedicated for `X.509` artifacts that should survive after `ECU` restart / shutdown.

2. "Volatile" (or "Session") is dedicated for `X.509` artifacts, that are valuable only in scope of current session of an application or `Functional Cluster`, importing these artifacts to the storage.

**[SWS_CRYPT_20007]**{DRAFT} ⌈The `X.509 Provider` shall store issued certificates in a persistent manner.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20009]**{DRAFT} ⌈When a certificates expires, the `X.509 Provider` shall replace the certificate with a new certificate. Additionally, the `X.509 Provider` may add the certificate on revocation list. The `X.509 Provider` shall update the internal state to reflect this change.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20010]**{DRAFT} ⌈`X.509 Provider` implementation shall require especial capability "Trust Master" from applications that will set specific certificate as a root of trust `ara::crypto::x509::X509Provider::SetAsRootOfTrust`.⌋*(RS_-CRYPTO_02306)*

**[SWS_CRYPT_20011]**{DRAFT} ⌈`X.509 Provider` shall support the Proof-Of-Possession (POP) of the private key.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40943]**{DRAFT} ⌈The functions `ara::crypto::x509::-X509Provider::ParseCustomCertExtensions`, `ara::crypto::x509:-:X509Provider::ParseCustomCertExtensions` shall parse the extension identified by the parameter oid of the provided Certificate and call the functions of the provided callback class customExtensionsParser in the order of occurence of the ASN.1 elements in the parsed certificate. If the parameter oid is not given, then ParseCustomCertExtensions shall parse all extensions of the certificate.

- If the parameter oid is given but the certificate does not contain an extension with the given oid, then ParseCustomCertExtensions shall return CryptoErrorDomain::kUnexpectedValue.

- If a function of the callback class customExtensionsParser returns any error, then ParseCustomCertExtensions shall stop parsing the certificate and return CryptoErrorDomain::kRuntimeFault.

⌋*(RS_CRYPTO_02306)*

### 7.4.3.1 Certificate Signing Request

**[SWS_CRYPT_20301]**{DRAFT} ⌈The `X.509 Provider` produces the Certificate Signing Request by `ara::crypto::x509::X509Provider::CreateCertSign-Request`. This is done in a specific context, which needs an identifier of the target asymmetric cryptographic algorithm and the corresponding public-private key pair. The `ara::crypto::x509::CertSignRequest`(CSR) is signed by the private key and contains the public key.⌋*(RS_CRYPTO_02306)* The identification of the used algorithm is done by the common name, as specified in 7.5.

The `X.509 Provider` delegates the `CSR` self-signature creation to the corresponding context, which is also responsible for processing of the correspondent private key.

**[SWS_CRYPT_20302]**{DRAFT} ⌈`X.509 Provider` shall encode all meta-information (`Distinguished name` and `X.509` Extensions). This meta-information is added during the `CSR` generation to the `CSR` before the signature is generated. The `Distinguished name` and `X.509` Extensions, can be either global or locally defined. The specific context is given either during the interface call (locally defined) or specified in the configuration (global). However, the specific local settings shall overwrite the global ones during the `CSR` generation. If no meta-information is provided, the global ones shall be used as default.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20303]**{DRAFT} ⌈All meta-information shall be encoded according to the `X.509` specification (as given in [32], [34], [35], [36], [37], [38], and [2]).⌋*(RS_-CRYPTO_02306)*

`X.509 Provider` distinguishes three states of a `CSR`:

1. "New" - the `CSR` is created, but is not yet sent to the Certification Authority (`CA`).

2. "Pending" - the `CSR` was already sent to the `CA`, but the internal was not yet updated. Either the `CSR` was not returned or was not processed.

3. "Retrieved" - the `CSR` was returned from the `CA`, and is either processed or the processing was not started yet.

When a signed `CSR` is retrieved, the `X.509 Provider` will import the `CSR` and starts the processing.

**[SWS_CRYPT_20304]**{DRAFT} ⌈Each `CSR` is an artifact produced by the `X.509 Provider` and is stored locally. The `CryptoAPI` provides an interface to allow an application or `Functional Cluster` to trigger the storing.⌋*(RS_CRYPTO_02306)*

### 7.4.3.2   Using Certificates

**[SWS_CRYPT_20601]**{DRAFT} **Importing** / **Installation** ⌈The `X.509 Provider` provides a mechanism for applications or `Functional Clusters` to import `ara::-crypto::x509::X509Provider::Import` or install certificates, parts of `certi-fication paths`, or full `certification paths`.⌋*(RS_CRYPTO_02306)*

This allows the user to integrate certificates into the system, especially when these are generated outside the system itself. Therefore, the `CryptoAPI` provides an interface to import certificates. This interface can be configured during the integration phase by using the `PortInterface`, as shown in 7.8, or the specific API call. When a certificate is imported, the `X.509 Provider` validates the certificate or the `certi-fication paths` with the corresponding `PKI`. Additionally, the `X.509 Provider` checks if all `Distinguished names` and `X.509` Extensions are matching the pre-configured meta-information (global information) or specified ones (local information). Specific meta-information is provided by the application or `Functional Cluster` via the interface call. If no specific meta-information is provided, the global ones are used as default. Importing can be done either via a file, which is stored on the system, or as an `ASN.1` encoded information directly. If an internal error occurred or the internal policy prohibits the importing, the caller will be informed by an error.

**[SWS_CRYPT_20602]**{DRAFT} **Exporting** ⌈The `X.509 Provider` exports a certificate, a bundle of certificates, a part of a `certification path`, or a full `certi-fication path`. The private key of the corresponding export is not included in the export.⌋*(RS_CRYPTO_02306)*

The export is done in `ASN.1` encoding according to `X.509` standard. The application or `Functional Cluster` can define the certificate format, such as `BER`, `DER`, or `PEM`, and specify if the export shall be stored as file or provided directly. The used meta-information, `Distinguished name` and `X.509` Extension, ca be provided locally during the export, or provided globally, as configured. However, the local ones will overwrite the global ones. If no meta-information is given, the global ones are used as default. Revoked certificated are not exported. In this case or the exporting cannot be done, either an internal error occurred or the internal policy prohibits it, the caller will be informed by an error.

**[SWS_CRYPT_20603]**{DRAFT} **Getting or Querying** ⌈When an application or a `Functional Cluster` needs a specific certificate, it can either use a configured one (this is provided via the `CryptoCertificateInterface` ) or can get a certificate via the `X.509 Provider` mechanism. If the user knows, which certificate it wnts to access, it can do this by providing the direct handle or the `COUID`. However, it occurs that the user does not know exactly which certificate is needed. Therefore, the `X.509 Provider` allows to query the certificate. The application or `Functional Cluster` then can provide either certificate information, such as `certificate serial number` or issuer, the meta-information, part of the meta-information, the environment the certificate is used for (e.g., `IPsec` or `TLS`), or provide parts of the `certification path`. In this case the `X.509 Provider` provides a list of all matching certificates `ara::crypto::x509::BasicCertInfo` or an error, when no matching certificate was found or the caller has not the corresponding access rights for the found certificates.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40912]**{DRAFT} **Querying with wildcards** ⌈When an instance of class `ara::crypto::x509::X509DN` is created all attributes of this instance shall be none-initialized. None-initialized attributes shall serve as wildcards.⌋*(RS_CRYPTO_-02306)*

**[SWS_CRYPT_40913]**{DRAFT} **Sets of certificates** ⌈The function `ara::crypto::x509::X509Provider::FindCertByDn` shall provide a set of all certificates that match the attributes of parameter subjectDn. The function findCertByDn shall ignore none-initialized attributes of parameter subjectDn for the search for certificates.⌋*(RS_CRYPTO_02306)*

Figure 7.8 shows the model elements that are relevant for the deployment of certificates.
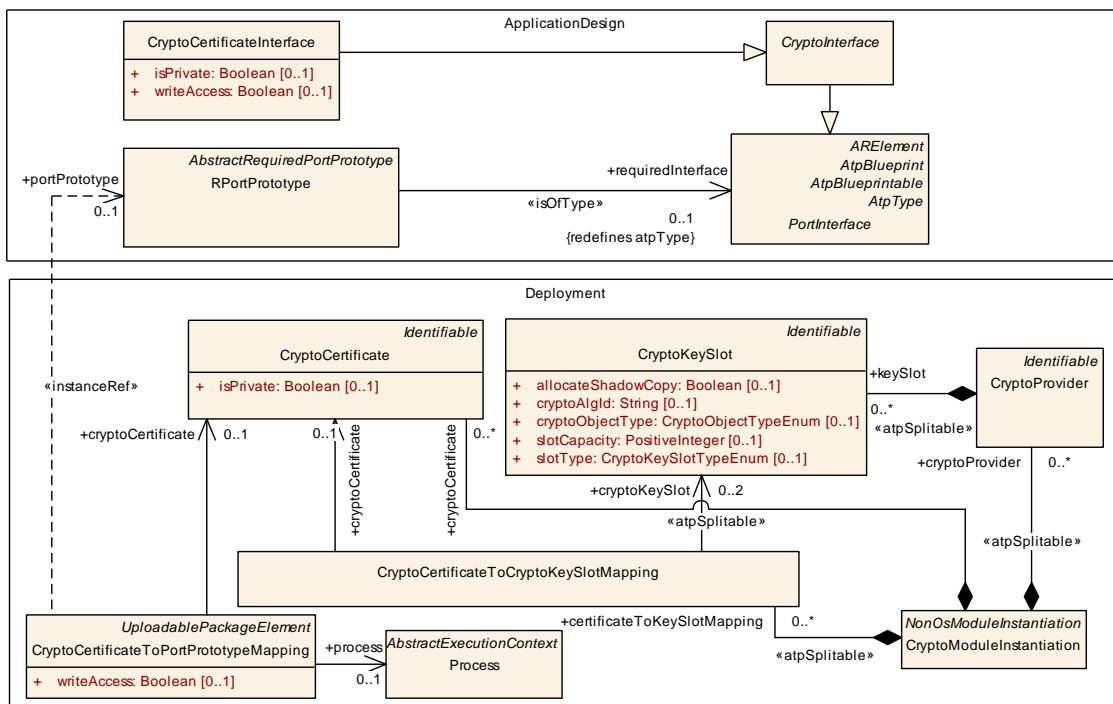


**Figure 7.8: Certificate deployment**

**[SWS_CRYPT_20611]**{DRAFT} **Validation of `certification path`** ⌈When a certificate is installed, the whole certificate chain must be validated based on the whole tree path up to the root certificate (e.g., vehicle root) by `ara::crypto::x509::X509Provider::CheckCertStatus`, `ara::crypto::x509::X509Provider::CheckCertStatus`, `ara::crypto::x509::X509Provider::CountCertsInChain`, `ara::crypto::x509::X509Provider::ParseCertChain`, `ara::crypto::x509::X509Provider::ParseCertChain`, `ara::crypto::x509::X509Provider::VerifyCertChain`. Only certificates, which are not root certificates, are checked.⌋*(RS_CRYPTO_02306)*

Root certificates are not checked, because these are the trust anchors of the system. Because root certificates play this special role, root certificate shall be stored in a tamper proof manner to avoid malicious manipulation. How this is done is not part of this standard.

**[SWS_CRYPT_20612]**{DRAFT} ⌈Supporting a full certificate life-cycle, the `FC Crypto` provides functionality to generate certificate signing request, where the needed encoding (i.e., `DER` or `PEM`) can be specified and the correct setting is ensured. The `CryptoAPI` provides this interface for CSR generation. Additionally, the `CryptoAPI` offers the specific interfaces to generate certificates and certificate chains, which can then be used by other protocols, i.e., `IKE`.⌋*(RS_CRYPTO_02306)*

The `PKI` contains the certificates of the vehicle side, i.e. all certificates or artifacts that are part of the vehicle. It is structured based on functions on the `CA` level (level 2) and on distributed issuers on the Sub-CA level (level 3). The top level is defined by the vehicle root certificate, which is provided by every OEM and serves as a trust anchor. Also `X.509 Provider` may keep root certificates of 3rd party trusted `CAs` in order to communicate with external service providers.

**[SWS_CRYPT_20613]**{DRAFT} ⌈The `FC Crypto` allows to encode and decode `ASN.1`-based standard formats (like [39, PKCS#8], [40, PKCS#12]), as specified in [41, X.680], [42, X.682], and [43, X.683]. The `CryptoAPI` allows an application or `Functional Cluster` to select the encoding.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20614]**{DRAFT} ⌈The `CryptoAPI` provide all required `X.509` functionality related with access to the certification target private key (used for signature of own certificate request, via top-level context interface). The target private key can have a type different from signature (e.g., decryption or key-agreement). This is specified by the connection between `CryptoCertificate` and `CryptoKeySlot`. This connection is done by a mapping.⌋*(RS_CRYPTO_02306)* The mapping is provided by `CryptoCertificateToCryptoKeySlotMapping` as shown in 7.8.

**[SWS_CRYPT_20615]**{DRAFT} ⌈The `X.509 Provider` shall verify self-signed certificates besides `PKI` based signatures. The `CryptoAPI` provides methods to specify the certificate and the used cryptographic algorithm. Based on the algorithm the `X.509 Provider` compares the given signature with the calculated one. If both are matching, the certificate is valid. Otherwise, the `X.509 Provider` will return an error.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20616]**{DRAFT} ⌈The access to the `PKI`-client's private key shall be used only internally and indirectly via the `X.509 Provider` interface. The private key will never leave the boundary of the `FC Crypto`.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20617]**{DRAFT} ⌈`X.509 Provider` is using the base cryptographic functions provided by the `Crypto Provider`. `CryptoAPI` provides related functions to store, retrieve, enumerate, verify, and use the information stored in the certificates.⌋ *(RS_CRYPTO_02306)*

**[SWS_CRYPT_20618]**{DRAFT} ⌈In the `CryptoAPI` context, the certificate store is protected from unauthorized access and tampering. This can be done by cryptographic mechanism, such as providing an `MAC`, or by storing the certificates in a secure storage, such as a `TPM`.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20619]**{DRAFT} ⌈During the initialization of the `FC Crypto`, all needed steps for service instantiation is done. This includes importing a root `CA` public key, setting up the `certification path` with all public keys along the path, checking the revocation status of certificates, updating the `X.509 Provider` internal management structure with certificate status, and the certificate ecosystem.⌋*(RS_CRYPTO_-02306)*

### 7.4.3.3 Revocation of certificates

The `X.509 Provider` supports the revocation of certificates. This is done by using standard mechanism, such as certificate revocation lists (`CRLs`) and certificate trust lists (`CTLs`). The `X.509 Provider` is the organizational part of the `FC Crypto`, which handles and stores during run-time these `CRLs` and `CTLs`. The `CryptoAPI` provides interfaces, which allow application and `Functional Clusters` to import, export, and manage these lists.

**[SWS_CRYPT_20901]**{DRAFT} **`CRL` and `CTL` usage** ⌈The `X.509 Provider` shall support `CRL` and `CTL`. The format of `CRL` and `CTL` are defined in [33, RFC 5280], [44, RFC 6518], [45, RFC 8398], and [46, RFC 8399] and is not part of this standard. The `X.509 Provider` can store the `CRL` and the `CTL` in an own internal used structure. However, the `X.509 Provider` can also use the provided information to update the corresponding elements. The update can be either the deletion of the element or setting a mark that the element was revoked.⌋*(RS_CRYPTO_02306)*

`CRL` is a list of digital certificates that have been revoked before their expiration date was reached. This list contains all the serial numbers of the revoked certificates and the revoked data.

**[SWS_CRYPT_20902]**{DRAFT} ⌈Given in [33] the `CRL` can contain two different states:

1. Revoked: certificates that are irreversibly revoked.

2. Hold: certificates that are marked as temporally invalid.

⌋*(RS_CRYPTO_02306)*

`CryptoAPI` shall provide two ways to get `CRL`:

1. Offline: An application or `Functional Cluster` provides a `CRL` to the `X.509 Provider`.

2. Online: `X.509 Provider` opens a secure channel to a backend system. After a successful established connection, the `X.509 Provider` gets the matching `CRL`. The location of the specific backend system can either configured or provided via an application or `Functional Cluster`.

**[SWS_CRYPT_20903]**{DRAFT} **Import** ⌈The `X.509 Provider` allows to import `ara::crypto::x509::X509Provider::ImportCrl` and update the `CRL`. These `CRL` can be either stored in the `X.509 Provider` separately or in combination with the certificate. The application or `Functional Cluster` can call the interface `ara::crypto::x509::X509Provider::ImportCrl`, which is provided by the `CryptoAPI`.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20904]**{DRAFT} ⌈The `X.509 Provider` shall support the online mode to get and update `CRL`.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20905]**{DRAFT} **Verify** ⌈The function `ara::crypto::x509::X509Provider::VerifyCert` shall verify if a certificate is valid. Therefore, the `X.509 Provider` checks additionally if a certificate was revoked, The revocation of the certificate is given via the `CRL`. This check can either be done via a call by an application or `Functional Cluster` (offline mode) or via a connection to a backend (online mode):

- In offline mode: An application or `Functional Cluster` provides the `CRL` to the `X.509 Provider` via an interface, which is exposed by the `CryptoAPI`.

- in online mode: The `X.509 Provider` uses a provided location to get the `CRL`. The location was provided by configuration or given in the interface call.

In both cases, the `X.509 Provider` uses the `CRL` to check if one of the internal stored certificate is listed. Is a certificate listed the `X.509 Provider` revokes the certificate internally.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20906]**{DRAFT} ⌈The `X.509 Provider` shall support the standard protocol, `ara::crypto::x509::OcspResponse` (as defined in [31, RFC 6960]) and `OCSP` Stapling (as defined in [47, RFC 6066], [48, RFC 6961], and [49, RFC 8446]), to check if a certificate is revoked. `OCSP` is an alternative to `CRLs`.

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20907]**{DRAFT} ⌈The `CryptoAPI` provides methods `ara::crypto::x509::X509Provider::CreateOcspRequest`, `ara::crypto::x509::X509Provider::CreateOcspRequest` to generate an `ara::crypto::x509::OcspRequest` request, which is defined in [31, RFC 6960]. The method can be used by an application or `Functional Cluster`.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20908]**{DRAFT} ⌈The `X.509 Provider` shall support request generation for the revocation of certificates.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_20909]**{DRAFT} **Signalization of revoked certificate by application or functional cluster** ⌈Dedicated applications are allowed to inform the `X.509 Provider` of a misuse or of the invalidity of certificates. The `X.509 Provider` stores this information by revoking internally the specified certificate. This can either be done in the internal structure where certificates are stored or by updating the stored revocation list. When the `X.509 Provider` generates a `CRL`, it uses its internal information.⌋ *(RS_CRYPTO_02306)*

**[SWS_CRYPT_20910]**{DRAFT} **Internal signalization of revoked certificate** ⌈The `X.509 Provider` shall mark certificates in its internal structure or update the stored revocation list as revoked, when the `X.509 Provider` recognizes that a certificate is not valid anymore and thus shall be revoked. This can occur during `certification path` validation or verification of a certificate.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40972]**{DRAFT} **Configuration Options** ⌈The FC Crypto shall provide two configuration options. A configuration field contains either the URL or the identifier to specify either the URL for the backend (local / stack usage) or the required service interface.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40973]**{DRAFT} ⌈The FC Crypto shall report an error by calling the online functions if the configuration is empty, not performed or the configuration parameter is not a matching combination. This allows the application to react on the existing problem.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40974]**{DRAFT} ⌈The FC Crypto shall handle the additional behavior to send or request OCSP tickets via the configured mechanism.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40975]**{DRAFT} ⌈The FC Crypto shall inform the application via an return value that the CRL was updated via configured mechanism.⌋*(RS_CRYPTO_-02306)*

**[SWS_CRYPT_40976]**{DRAFT} ⌈The FC Crypto shall inform the application via an return value, that the online information was sent to the configured mechanism.⌋*(RS_-CRYPTO_02306)*

**[SWS_CRYPT_40977]**{DRAFT} ⌈The FC Crypto shall inform the application via an return value, that the validity of a given certificate was checked online via the configured mechanism.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40978]**{DRAFT} ⌈The FC Crypto shall provide a mechanism, which is used via the calling user (FC Crypto itself or application), allowing to get the last received OCSP ticket.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40979]**{DRAFT} ⌈The FC Crypto shall update the internal state of a certificate, if the certificate was invalidated via a given OCSP ticket and if the certificate is handled via the internal certificates management.⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40980]**{DRAFT} ⌈The FC Crypto shall point to the successor of an invalidated certificate, if the OCSP ticket provides a successor.⌋*(RS_CRYPTO_02306)*

## 7.5 Cryptographic Primitives Naming Convention

`Crypto Providers` transforms the specific needed algorithm, which was configured during integration phase, into the by `FC Crypto` provided vendor specific algorithm. Supporting this decoupling of configuration from instantiation and enabling the support of future upcoming cryptographic algorithm, this specification does not provide a concrete list of cryptographic algorithms' identifiers and does not suppose usage of numerical identifiers. Instead of this, the vendor shall provide string names of supported algorithms in accompanying documentation.

The string names are used for the following:

- They are used as parameters by interface functions of a `Crypto Provider`.

- They serve as identifiers to cryptographic algorithms.

- The `Crypto Provider` interprets the string names and matches it to the algorithm, which is provided by `FC Crypto`.

**[SWS_CRYPT_03910]**{DRAFT} **Configuration format for cryptographic algorithms** ⌈The string names to identify cryptographic algorithms shall satisfy the following rules:

1. The string names contains only Latin alphanumeric characters.

2. The string names contain up to 6 delimiters for cryptographic algorithm definition.

3. The string names is case insensitive. Thus, all comparisons of the identifiers shall be always case insensitive.

4. The string names to identify cryptographic algorithms shall satisfy the following structures:

   `"{TargetTransformation(Mode)} / {SupportingAlgorithms} /`
   `                    {Encoding&Padding}"`

where

- `"{TargetTransformation(Mode)}"` – a specifier of target transformation: for complex transformations it is a mode name, but for fully-defined algorithms it is just their name.

- `"{SupportingAlgorithms}"` – a specifier of basic cryptographic algorithm(s) including key length andor block length.

- `"{Encoding&Padding}"` – a specifier of encoding and/or padding method. It can support following predefined name (equal to empty specification):

  - `"Zero"` – a default encoding & padding method: if data are already aligned to the block boundary then it doesn't add anything, but if they are not aligned then applies a padding by `'\0'` bytes up to the block boundary.

Allowed delimiters:

- `'/'` – separator between main components of the whole algorithm specification.

- `'_'` – separator instead of general separation characters (e.g.: `' '`, `'.'`, `':'`, `'-'`, `''`) in original name of standard. This delimiter can be applied between two digits or two letters only!

- `'-'` – separator between a base algorithm name and its precise specifiers that define key-length or block-length in bits.

- `'+'` – separator between a few base algorithms' specifications for a cascade transformation definition.

- `','` – separator between a few base algorithms' specifications for a case if the whole algorithm is based on a few types of basic transformations.

- `'.'` – separator between a common name of a standard and its specific part or its version that precises a specification of concrete transformation.

⌋*(RS_CRYPTO_02308)*

Examples of well-known algorithm names: `"ECDSA-256"`, `"ECDH-256"`, `"AES-128"`, `"Camellia-256"`, `"3DES-168"`, `"ChaCha20"`, `"GOST28147_89"`, `"SHA1"`, `"SHA2-256"`, `"GOSTR3410.94"`, `"GOSTR3410.2001"`, `"GOSTR3410.2012-512"`.

Examples of well-known modes names: `"ECB"`, `"OFB"`, `"CFB"`, `"CBC"`, `"PCBC"`, `"CTR"`, `"HMAC"`, `"CBC_MAC"`, `"OMAC1"`, `"OMAC2"`, `"VMAC"`, `"Poly1305"`, `"CCM"`, `"GCM"`, `"OCB"`, `"CWC"`, `"EAX"`, `"KDF1"`, `"KDF2"`, `"KDF3"`, `"MGF1"`.

Examples of the encoding and padding names: `"ANSI_X923"`, `"ISO10126"`, `"PKCS7"`, `"ISO_IEC7816_4"`, `"PKCS1.v1_5"`, `"OAEP"`, `"OAEPplus"`, `"SAEP"`, `"SAEPplus"`, `"PSS"`, `"EME"`, `"EMSA"`.

Examples of fully defined transformations:

- `"ECDSA-384"` means ECDSA signature algorithm with private key-length 384 bit.

- `"ECDH-512"` means ECDH key agreement algorithm with private key-length 512 bit.

- `"CTRAES-256"` means a CTR-mode stream cipher based on AES algorithm with key-length 256 bit.

- `"CBCAES-192+Camellia-192/PKCS7"` means CBC-mode cipher based on cascade application of AES-192 and Camellia-192 with padding of last block according to PKCS#7.

- `"HMACSHA-256"` means HMAC based on SHA-256.

If an algorithm support a few variable length parameters then they shall be specified in following order:

key, IO-block or output digest, IV or input block (e.g.: `"Kalyna-512-256"` means block cipher Kalina with 512-bit key and 256-bit block).

If a transformation is based on a few basic cryptographic algorithms then they shall be specified in an order corresponding to the level of their application (see example below for RSA).

Following Mode specifications can be used for RSA-based algorithms:

- `"SIG"` – signature primitive (e.g., `"SIGRSA-2048,SHA-160PKCS1.v1_5,EMSA"`)

- `"VER"` – verification primitive (e.g., `"VERRSA-2048,SHA-160PKCS1.v1_5,EMSA"`)

- `"ENC"` – encryption primitive (e.g., `"ENCRSA-2048,MGF1,SHA-160PKCS1.v1_5,EME"`, `"ENCRSA-4096,MGF1,SHA2-256OAEP,EME"`)

- `"DEC"` – decryption primitive (e.g., `"DECRSA-2048,MGF1,SHA-160PKCS1.v1_5,EME"`, `"DECRSA-4096,MGF1,SHA2-256OAEP,EME"`)

- `"KEM"` – Key Encapsulation Mechanism (e.g., `"KEM/RSA-2048,AES-128,KDF3,SHA-256"`)

A supplier should strive to use shortest names of algorithms, sufficient for their unambiguous identification.

# 8 API specification

## 8.1 C++ language binding Crypto Provider

**[SWS_CRYPT_20100]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | AuthCipherCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::AuthCipherCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" |
| Description: | Generalized Authenticated Cipher Context interface. Methods of the derived interface Buffered Digest are used for authentication of associated public data. Methods of the derived interface StreamCipherCtx are used for encryption/decryption and authentication of confidential part of message. The data processing must be executed in following order: |
| | Call one of the Start() methods. Process all associated public data via calls of Update() methods. Process the confidential part of the message via calls of ProcessBlocks(), Process Bytes() (and optionally FinishBytes()) methods. Call the Finish() method due to finalize the authentication code calculation (and get it optionally). Copy of the calculated MAC may be extracted (by GetDigest()) or compared internally (by Compare()). Receiver side should not use decrypted data before finishing of the whole decryption and authentication process! I.e. decrypted data can be used only after successful MAC verification! |

⌋*(RS_CRYPTO_02207)*

**[SWS_CRYPT_29030]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | BlockService |
| Scope: | namespace ara::crypto::cryp |
| Base class: | ExtensionService |
| Syntax: | `class ara::crypto::cryp::BlockService :  public ExtensionService {...};` |
| Header file: | #include "ara/crypto/cryp/block_service.h" |
| Description: | Extension meta-information service for block cipher contexts. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_20400]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CryptoContext |
| Scope: | namespace ara::crypto::cryp |
| Syntax: | `class ara::crypto::cryp::CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/crypto_context.h" |
| Description: | A common interface of a mutable cryptographic context, i.e. that is not binded to a single crypto object. |

⌋*(RS_CRYPTO_02008)*

### [SWS_CRYPT_20500]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CryptoObject |
| Scope: | namespace ara::crypto::cryp |
| Syntax: | `class ara::crypto::cryp::CryptoObject {...};` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | A common interface for all cryptograhic objects recognizable by the Crypto Provider. This interface (or any its derivative) represents a non-mutable (after completion) object loadable to a temporary transformation context. |

⌋*(RS_CRYPTO_02005)*

### [SWS_CRYPT_20600]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CryptoPrimitiveId |
| Scope: | namespace ara::crypto::cryp |
| Syntax: | `class ara::crypto::cryp::CryptoPrimitiveId {...};` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" |
| Description: | Common interface for identification of all Crypto Primitives and their keys & parameters. |

⌋*(RS_CRYPTO_02005)*

### [SWS_CRYPT_20700]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CryptoProvider |
| Scope: | namespace ara::crypto::cryp |
| Syntax: | `class ara::crypto::cryp::CryptoProvider {...};` |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" |
| Description: | Crypto Provider is a "factory" interface of all supported Crypto Primitives and a "trusted environmet" for internal communications between them. All Crypto Primitives should have an actual reference to their parent Crypto Provider. A Crypto Provider can be destroyed only after destroying of all its daughterly Crypto Primitives. Each method of this interface that creates a Crypto Primitive instance is non-constant, because any such creation increases a references counter of the Crypto Primitive. |

⌋*(RS_CRYPTO_02305, RS_CRYPTO_02307, RS_CRYPTO_02401)*

### [SWS_CRYPT_29020]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CryptoService |
| Scope: | namespace ara::crypto::cryp |
| Base class: | ExtensionService |
| Syntax: | `class ara::crypto::cryp::CryptoService :  public ExtensionService {...};` |
| Header file: | #include "ara/crypto/cryp/crypto_service.h" |
| Description: | Extension meta-information service for cryptographic contexts. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_20800]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | DecryptorPrivateCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::DecryptorPrivateCtx : public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/decryptor_private_ctx.h" |
| Description: | Asymmetric Decryption Private key Context interface. |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_29010]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | DigestService |
| Scope: | namespace ara::crypto::cryp |
| Base class: | BlockService |
| Syntax: | `class ara::crypto::cryp::DigestService : public BlockService {...};` |
| Header file: | #include "ara/crypto/cryp/digest_service.h" |
| Description: | Extension meta-information service for digest producing contexts. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_21000]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | EncryptorPublicCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::EncryptorPublicCtx : public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/encryptor_public_ctx.h" |
| Description: | Asymmetric Encryption Public key Context interface. |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_29040]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | ExtensionService |
| Scope: | namespace ara::crypto::cryp |
| Syntax: | `class ara::crypto::cryp::ExtensionService {...};` |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Basic meta-information service for all contexts. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_21100]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | HashFunctionCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::HashFunctionCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/hash_function_ctx.h" |
| Description: | Hash function interface. |

⌋*(RS_CRYPTO_02205)*

**[SWS_CRYPT_21300]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | KeyAgreementPrivateCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::KeyAgreementPrivateCtx :  public Crypto Context {...};` |
| Header file: | #include "ara/crypto/cryp/key_agreement_private_ctx.h" |
| Description: | Key Agreement Private key Context interface (Diffie Hellman or conceptually similar). |

⌋*(RS_CRYPTO_02104)*

**[SWS_CRYPT_21400]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | KeyDecapsulatorPrivateCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::KeyDecapsulatorPrivateCtx :  public Crypto Context {...};` |
| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" |
| Description: | Asymmetric Key Encapsulation Mechanism (KEM) Private key Context interface. |

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02209)*

**[SWS_CRYPT_21500]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | KeyDerivationFunctionCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::KeyDerivationFunctionCtx :  public Crypto Context {...};` |
| Header file: | #include "ara/crypto/cryp/key_derivation_function_ctx.h" |
| Description: | Key Derivation Function interface. |

⌋*(RS_CRYPTO_02103)*

**[SWS_CRYPT_21800]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | KeyEncapsulatorPublicCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::KeyEncapsulatorPublicCtx :  public Crypto Context {...};` |
| Header file: | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" |
| Description: | Asymmetric Key Encapsulation Mechanism (KEM) Public key Context interface. |

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02209)*

**[SWS_CRYPT_22100]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | MessageAuthnCodeCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::MessageAuthnCodeCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" |
| Description: | Keyed Message Authentication Code Context interface definition (MAC/HMAC). |

⌋*(RS_CRYPTO_02203)*

**[SWS_CRYPT_22200]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | MsgRecoveryPublicCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::MsgRecoveryPublicCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" |
| Description: | A public key context for asymmetric recovery of a short message and its signature verification (RSA-like). Restricted groups of trusted subscribers can use this primitive for simultaneous provisioning of confidentiality, authenticity and non-repudiation of short messages, if the public key is generated appropriately and kept in secret. If (0 == BlockCryptor::ProcessBlock(...)) then the input message-block is violated. |

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204)*

**[SWS_CRYPT_22500]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | PrivateKey |
| Scope: | namespace ara::crypto::cryp |
| Base class: | RestrictedUseObject |
| Syntax: | `class ara::crypto::cryp::PrivateKey :  public RestrictedUseObject {...};` |

▽

△

| | |
|---|---|
| **Header file:** | #include "ara/crypto/cryp/cryobj/private_key.h" |
| **Description:** | Generalized Asymmetric Private Key interface. |

⌋*(RS_CRYPTO_02002, RS_CRYPTO_02403)*

## [SWS_CRYPT_22700]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | class |
| **Symbol:** | PublicKey |
| **Scope:** | namespace ara::crypto::cryp |
| **Base class:** | RestrictedUseObject |
| **Syntax:** | `class ara::crypto::cryp::PublicKey :  public RestrictedUseObject {...};` |
| **Header file:** | #include "ara/crypto/cryp/cryobj/public_key.h" |
| **Description:** | General Asymmetric Public Key interface. |

⌋*(RS_CRYPTO_02202)*

## [SWS_CRYPT_22900]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | class |
| **Symbol:** | RandomGeneratorCtx |
| **Scope:** | namespace ara::crypto::cryp |
| **Base class:** | CryptoContext |
| **Syntax:** | `class ara::crypto::cryp::RandomGeneratorCtx :  public CryptoContext {...};` |
| **Header file:** | #include "ara/crypto/cryp/random_generator_ctx.h" |
| **Description:** | Interface of Random Number Generator Context. |

⌋*(RS_CRYPTO_02206)*

## [SWS_CRYPT_24800]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | class |
| **Symbol:** | RestrictedUseObject |
| **Scope:** | namespace ara::crypto::cryp |
| **Base class:** | CryptoObject |
| **Syntax:** | `class ara::crypto::cryp::RestrictedUseObject :  public CryptoObject {...};` |
| **Header file:** | #include "ara/crypto/cryp/cryobj/restricted_use_object.h" |
| **Description:** | A common interface for all objects supporting the usage restriction. |

⌋*(RS_CRYPTO_02008)*

## [SWS_CRYPT_23000]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | class |
| **Symbol:** | SecretSeed |
| **Scope:** | namespace ara::crypto::cryp |

▽

△

| | |
|---|---|
| ***Base class:*** | RestrictedUseObject |
| ***Syntax:*** | `class ara::crypto::cryp::SecretSeed :  public RestrictedUseObject {...};` |
| ***Header file:*** | #include "ara/crypto/cryp/cryobj/secret_seed.h" |
| ***Description:*** | Secret Seed object interface. This object contains a raw bit sequence of specific length (without any filtering of allowed/disallowed values)! The secret seed value can be loaded only to a non-key input of a cryptographic transformation context (like IV/salt/nonce)! Bit length of the secret seed is specific to concret crypto algorithm and corresponds to maximum of its input/ output/salt block-length. |

⌋*(RS_CRYPTO_02007)*

## **[SWS_CRYPT_23200]**{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | class |
| ***Symbol:*** | SigEncodePrivateCtx |
| ***Scope:*** | namespace ara::crypto::cryp |
| ***Base class:*** | CryptoContext |
| ***Syntax:*** | `class ara::crypto::cryp::SigEncodePrivateCtx :  public CryptoContext {...};` |
| ***Header file:*** | #include "ara/crypto/cryp/sig_encode_private_ctx.h" |
| ***Description:*** | A private key context for asymmetric signature calculation and short message encoding (RSA-like). Restricted groups of trusted subscribers can use this primitive for simultaneous provisioning of confidentiality, authenticity and non-repudiation of short messages, if the public key is generated appropriately and kept in secret. |

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204)*

## **[SWS_CRYPT_29000]**{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | class |
| ***Symbol:*** | SignatureService |
| ***Scope:*** | namespace ara::crypto::cryp |
| ***Base class:*** | ExtensionService |
| ***Syntax:*** | `class ara::crypto::cryp::SignatureService :  public ExtensionService {...};` |
| ***Header file:*** | #include "ara/crypto/cryp/signature_service.h" |
| ***Description:*** | Extension meta-information service for signature contexts. |

⌋*(RS_CRYPTO_02309)*

## **[SWS_CRYPT_23300]**{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | class |
| ***Symbol:*** | Signature |
| ***Scope:*** | namespace ara::crypto::cryp |
| ***Base class:*** | CryptoObject |
| ***Syntax:*** | `class ara::crypto::cryp::Signature :  public CryptoObject {...};` |
| ***Header file:*** | #include "ara/crypto/cryp/cryobj/signature.h" |

▽

△

| Description: | Signature container interface This interface is applicable for keeping the Digital Signature, Hash Digest, (Hash-based) Message Authentication Code (MAC/HMAC). In case of a keyed signature (Digital Signature or MAC/HMAC) a COUID of the signature verification key can be obtained by a call of CryptoObject::HasDependence()! |
|---|---|

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02204, RS_CRYPTO_02205)*

## [SWS_CRYPT_23500]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | SignerPrivateCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::SignerPrivateCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/signer_private_ctx.h" |
| Description: | Signature Private key Context interface. |

⌋*(RS_CRYPTO_02204)*

## [SWS_CRYPT_23600]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | StreamCipherCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::StreamCipherCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" |
| Description: | Generalized Stream Cipher Context interface (it covers all modes of operation). |

⌋*(RS_CRYPTO_02201)*

## [SWS_CRYPT_23700]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | SymmetricBlockCipherCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::SymmetricBlockCipherCtx :  public Crypto Context {...};` |
| Header file: | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" |
| Description: | Interface of a Symmetric Block Cipher Context with padding. |

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_23800]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | SymmetricKey |
| Scope: | namespace ara::crypto::cryp |
| Base class: | RestrictedUseObject |
| Syntax: | `class ara::crypto::cryp::SymmetricKey :  public RestrictedUseObject {...};` |
| Header file: | #include "ara/crypto/cryp/cryobj/symmetric_key.h" |
| Description: | Symmetric Key interface. |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02403)*

**[SWS_CRYPT_24000]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | SymmetricKeyWrapperCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::SymmetricKeyWrapperCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" |
| Description: | Context of a symmetric key wrap algorithm (for AES it should be compatible with RFC3394 or RFC5649). The public interface of this context is dedicated for raw key material wrapping/ unwrapping, i.e. without any meta-information assigned to the key material in source crypto object. But additionally this context type should support some "hidden" low-level methods suitable for whole crypto object exporting/importing. Key Wrapping of a whole crypto object (including associated meta-information) can be done by methods: ExportSecuredObject() and ImportSecuredObject(), but without compliance to RFC3394 or RFC5649. |

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02208)*

**[SWS_CRYPT_24100]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | VerifierPublicCtx |
| Scope: | namespace ara::crypto::cryp |
| Base class: | CryptoContext |
| Syntax: | `class ara::crypto::cryp::VerifierPublicCtx :  public CryptoContext {...};` |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" |
| Description: | Signature Verification Public key Context interface. |

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_20319]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Check(const Signature &expected) |
| Scope: | class ara::crypto::cryp::AuthCipherCtx |

▽

$\triangle$

| Syntax: | virtual ara::core::Result<bool> ara::crypto::cryp::AuthCipher Ctx::Check (const Signature &expected) const noexcept=0; | |
|---|---|---|
| Parameters (in): | expected | the signature object containing an expected digest value |
| Return value: | ara::core::Result< bool > | true if value and meta-information of the provided "signature" object is identical to calculated digest and current configuration of the context respectively; but false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |
| | CryptoErrorDomain::kIncompatible Object | if the provided "signature" object was produced by another crypto primitive type |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| Description: | Check the calculated digest against an expected "signature" object. Entire digest value is kept in the context up to next call Start(), therefore it can be verified again or extracted. This method can be implemented as "inline" after standartization of function ara::core::memcmp(). | |

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02204)*

## [SWS_CRYPT_20102]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetDigestService() |
| Scope: | class ara::crypto::cryp::AuthCipherCtx |
| Syntax: | virtual DigestService::Uptr ara::crypto::cryp::AuthCipherCtx::Get DigestService () const noexcept=0; |
| Return value: | DigestService::Uptr | – |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" |
| Description: | Get DigestService instance. |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_20316]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetDigest(std::size_t offset=0) | |
| Scope: | class ara::crypto::cryp::AuthCipherCtx | |
| Syntax: | virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::AuthCipherCtx::GetDigest (std::size_t offset=0) const noexcept=0; | |
| Parameters (in): | offset | position of the first byte of digest that should be placed to the output buffer |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | an output buffer storing the requested digest fragment or the full digest |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |

$\triangledown$

△

| | |
|---|---|
| | CryptoErrorDomain::kUsageViolation | if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context initialized by a key without kAllow Signature permission |
| *Header file:* | #include "ara/crypto/cryp/auth_cipher_ctx.h" |
| *Description:* | Retrieve the calculated digest. The entire digest value is kept in the context until the next call of Start(). Therefore, the digest can be re-checked or extracted at any time. If the offset is larger than the digest, an empty buffer shall be returned. This method can be implemented as "inline" after standardization of function ara::core::memcpy(). |

⌋*(RS_CRYPTO_02207)*

## [SWS_CRYPT_21715]{DRAFT} ⌈

| | | |
|---|---|---|
| *Kind:* | function | |
| *Symbol:* | GetTransformation() | |
| *Scope:* | class ara::crypto::cryp::AuthCipherCtx | |
| *Syntax:* | `virtual ara::core::Result<CryptoTransform> ara::crypto::cryp::Auth CipherCtx::GetTransformation () const noexcept=0;` | |
| *Return value:* | ara::core::Result< CryptoTransform > | CryptoTransform |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Errors:* | CryptoErrorDomain::kUninitialized Context | if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet |
| *Header file:* | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| *Description:* | Get the kind of transformation configured for this context: kEncrypt or kDecrypt. | |

⌋*(RS_CRYPTO_02309)*

## [SWS_CRYPT_20103]{DRAFT} ⌈

| | | |
|---|---|---|
| *Kind:* | function | |
| *Symbol:* | GetMaxAssociatedDataSize() | |
| *Scope:* | class ara::crypto::cryp::AuthCipherCtx | |
| *Syntax:* | `virtual std::uint64_t ara::crypto::cryp::AuthCipherCtx::GetMax AssociatedDataSize () const noexcept=0;` | |
| *Return value:* | std::uint64_t | maximal supported size of associated public data in bytes |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| *Description:* | Get maximal supported size of associated public data. | |

⌋*(RS_CRYPTO_02309)*

## [SWS_CRYPT_23634]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | ProcessConfidentialData(ReadOnlyMemRegion in, ara::core::Optional< ReadOnlyMemRegion > expectedTag) |

▽

△

| Scope: | class ara::crypto::cryp::AuthCipherCtx |
|---|---|
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData (ReadOnlyMem Region in, ara::core::Optional< ReadOnlyMemRegion > expectedTag) noexcept=0;` |
| Parameters (in): | in | the input buffer containing the full message |
| | expectedTag | optional pointer to read only mem region containing the auth-tag for verification. |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | the processed data |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if size of the input buffer is not divisible by the block size (see GetBlockSize()) |
| | CryptoErrorDomain::kProcessingNot Started | if the data processing was not started by a call of the Start() method |
| | CryptoErrorDomain::kAuthTagNotValid | if the processed data cannot be authenticated |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| Description: | Process confidential data and return result. The input buffer will be overwritten by the processed message. This function is the final call, i.e. all associated data must have been already provided. Hence, the function will check the authentication tag and only return the processed data, if the tag is valid. | |

⌊*(RS_CRYPTO_02302)*

## [SWS_CRYPT_23635]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ProcessConfidentialData(ReadWriteMemRegion inOut, ara::core::Optional< ReadOnlyMem Region > expectedTag) |
| Scope: | class ara::crypto::cryp::AuthCipherCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipher Ctx::ProcessConfidentialData (ReadWriteMemRegion inOut, ara::core::Optional< ReadOnlyMemRegion > expectedTag) noexcept=0;` |
| Parameters (in): | inOut | the input buffer containing the full message |
| | expectedTag | optional pointer to read only mem region containing the auth-tag for verification. |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if size of the input buffer is not divisible by the block size (see GetBlockSize()) |
| | CryptoErrorDomain::kProcessingNot Started | if the data processing was not started by a call of the Start() method |
| | CryptoErrorDomain::kAuthTagNotValid | if the processed data cannot be authenticated |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| Description: | Process confidential data and update the input buffer with the processed message. The input buffer will be overwritten by the processed message After this method is called no additional associated data may be updated. | |

⌊*(RS_CRYPTO_02302)*

**[SWS_CRYPT_20414]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Reset() |
| Scope: | class ara::crypto::cryp::AuthCipherCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipher Ctx::Reset () noexcept=0;` |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" |
| Description: | Clear the crypto context. . |

⌋*(RS_CRYPTO_02108)*

**[SWS_CRYPT_23911]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt) | |
| Scope: | class ara::crypto::cryp::AuthCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipherCtx::Set Key (const SymmetricKey &key, CryptoTransform transform=Crypto Transform::kEncrypt) noexcept=0;` | |
| Parameters (in): | key | the source key object |
| | transform | the "direction" indicator: deploy the key for encryption (if true) or for decryption (if false) |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| Description: | Set (deploy) a key to the authenticated cipher symmetric algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

**[SWS_CRYPT_24714]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Start(ReadOnlyMemRegion iv=ReadOnlyMemRegion()) | |
| Scope: | class ara::crypto::cryp::AuthCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipher Ctx::Start (ReadOnlyMemRegion iv=ReadOnlyMemRegion()) noexcept=0;` | |
| Parameters (in): | iv | an optional Initialization Vector (IV) or "nonce" value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |

▽

△

| Errors: | CryptoErrorDomain::kUninitialized Context | if the context was not initialized |
| --- | --- | --- |
| | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false) |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| Description: | Initialize the context for a new data processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_24715]{DRAFT} ⌈

| Kind: | function | |
| --- | --- | --- |
| Symbol: | Start(const SecretSeed &iv) | |
| Scope: | class ara::crypto::cryp::AuthCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipher Ctx::Start (const SecretSeed &iv) noexcept=0;` | |
| Parameters (in): | iv | the Initialization Vector (IV) or "nonce" object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if the context was not initialized |
| | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation |
| | CryptoErrorDomain::kUsageViolation | if this transformation type is prohibited by the "allowed usage" restrictions of the provided Secret Seed object |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| Description: | Initialize the context for a new data processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_20312]{DRAFT} ⌈

| Kind: | function | |
| --- | --- | --- |
| Symbol: | UpdateAssociatedData(const RestrictedUseObject &in) | |
| Scope: | class ara::crypto::cryp::AuthCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipher Ctx::UpdateAssociatedData (const RestrictedUseObject &in) noexcept=0;` | |
| Parameters (in): | in | a part of input message that should be processed |
| Return value: | ara::core::Result< void > | – |

▽

△

| | |
|---|---|
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| | CryptoErrorDomain::kInvalidUsage Order | if ProcessConfidentialData has already been called |
| **Header file:** | #include "ara/crypto/cryp/auth_cipher_ctx.h" |
| **Description:** | Update the digest calculation by the specified RestrictedUseObject. This method is dedicated for cases then the RestrictedUseObject is a part of the "message". |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_20313]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | UpdateAssociatedData(ReadOnlyMemRegion in) | |
| **Scope:** | class ara::crypto::cryp::AuthCipherCtx | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipher Ctx::UpdateAssociatedData (ReadOnlyMemRegion in) noexcept=0;` | |
| **Parameters (in):** | in | a part of the input message that should be processed |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| | CryptoErrorDomain::kInvalidUsage Order | if ProcessConfidentialData has already been called |
| **Header file:** | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| **Description:** | Update the digest calculation by a new chunk of associated data. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_20314]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | UpdateAssociatedData(std::uint8_t in) | |
| **Scope:** | class ara::crypto::cryp::AuthCipherCtx | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::AuthCipher Ctx::UpdateAssociatedData (std::uint8_t in) noexcept=0;` | |
| **Parameters (in):** | in | a byte value that is a part of input message |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| | CryptoErrorDomain::kInvalidUsage Order | if ProcessConfidentialData has already been called |
| **Header file:** | #include "ara/crypto/cryp/auth_cipher_ctx.h" | |
| **Description:** | Update the digest calculation by the specified Byte. This method is convenient for processing of constant tags. | |

⌋*(RS_CRYPTO_02302)*

**[SWS_CRYPT_29035]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetActualIvBitLength(ara::core::Optional< CryptoObjectUid > ivUid) |
| **Scope:** | class ara::crypto::cryp::BlockService |
| **Syntax:** | `virtual std::size_t ara::crypto::cryp::BlockService::GetActualIvBit`<br>`Length (ara::core::Optional< CryptoObjectUid > ivUid) const`<br>`noexcept=0;` |
| **Parameters (in):** | ivUid | optional pointer to a buffer for saving an COUID of a IV object now loaded to the context. If the context was initialized by a SecretSeed object then the output buffer *ivUid must be filled by COUID of this loaded IV object, in other cases *ivUid must be filled by all zeros. |
| **Return value:** | std::size_t | actual length of the IV (now set to the algorithm context) in bits |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/block_service.h" | |
| **Description:** | Get actual bit-length of an IV loaded to the context. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29033]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetBlockSize() |
| **Scope:** | class ara::crypto::cryp::BlockService |
| **Syntax:** | `virtual std::size_t ara::crypto::cryp::BlockService::GetBlockSize ()`<br>`const noexcept=0;` |
| **Return value:** | std::size_t | size of the block in bytes |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/block_service.h" | |
| **Description:** | Get block (or internal buffer) size of the base algorithm. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29032]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetIvSize() |
| **Scope:** | class ara::crypto::cryp::BlockService |
| **Syntax:** | `virtual std::size_t ara::crypto::cryp::BlockService::GetIvSize ()`<br>`const noexcept=0;` |
| **Return value:** | std::size_t | default expected size of IV in bytes |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/block_service.h" | |
| **Description:** | Get default expected size of the Initialization Vector (IV) or nonce. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29034]**{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | IsValidIvSize(std::size_t ivSize) |
| **Scope:** | class ara::crypto::cryp::BlockService |
| **Syntax:** | `virtual bool ara::crypto::cryp::BlockService::IsValidIvSize` `(std::size_t ivSize) const noexcept=0;` |
| **Parameters (in):** | ivSize | the length of the IV in bytes |
| **Return value:** | bool | true if provided IV length is supported by the algorithm and false otherwise |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/block_service.h" | |
| **Description:** | Verify validity of specific Initialization Vector (IV) length. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_20401]**{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | ~CryptoContext() |
| **Scope:** | class ara::crypto::cryp::CryptoContext |
| **Syntax:** | `virtual ara::crypto::cryp::CryptoContext::~CryptoContext ()` `noexcept=default;` |
| **Exception Safety:** | noexcept |
| **Header file:** | #include "ara/crypto/cryp/crypto_context.h" |
| **Description:** | Destructor. |

⌋*(RS_CRYPTO_02008)*

**[SWS_CRYPT_20411]**{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | GetCryptoPrimitiveId() |
| **Scope:** | class ara::crypto::cryp::CryptoContext |
| **Syntax:** | `virtual CryptoPrimitiveId::Uptr ara::crypto::cryp::CryptoContext::Get` `CryptoPrimitiveId () const noexcept=0;` |
| **Return value:** | CryptoPrimitiveId::Uptr | – |
| **Exception Safety:** | noexcept | |
| **Header file:** | #include "ara/crypto/cryp/crypto_context.h" | |
| **Description:** | Return CryptoPrimitivId instance containing instance identification. | |

⌋*(RS_CRYPTO_02008)*

**[SWS_CRYPT_20412]**{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | IsInitialized() |
| **Scope:** | class ara::crypto::cryp::CryptoContext |

▽

△

| Syntax: | virtual bool ara::crypto::cryp::CryptoContext::IsInitialized () const noexcept=0; | |
|---|---|---|
| Return value: | bool | true if the crypto context is completely initialized and ready to use, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/crypto_context.h" | |
| Description: | Check if the crypto context is already initialized and ready to use. It checks all required values, including: key value, IV/seed, etc. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_30214]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const CryptoContext &other) | |
| Scope: | class ara::crypto::cryp::CryptoContext | |
| Syntax: | CryptoContext& ara::crypto::cryp::CryptoContext::operator= (const CryptoContext &other)=default; | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoContext & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/crypto_context.h" | |
| Description: | Copy-assign another CryptoContext to this instance. | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_30215]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(CryptoContext &&other) | |
| Scope: | class ara::crypto::cryp::CryptoContext | |
| Syntax: | CryptoContext& ara::crypto::cryp::CryptoContext::operator= (CryptoContext &&other)=default; | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoContext & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/crypto_context.h" | |
| Description: | Move-assign another CryptoContext to this instance. | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_20654]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | MyProvider() | |
| Scope: | class ara::crypto::cryp::CryptoContext | |
| Syntax: | virtual CryptoProvider& ara::crypto::cryp::CryptoContext::MyProvider () const noexcept=0; | |
| Return value: | CryptoProvider & | a reference to Crypto Provider instance that provides this context |

▽

$\triangle$

| | |
|---|---|
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/cryp/crypto_context.h" |
| **Description:** | Get a reference to Crypto Provider of this context. |

⌋*(RS_CRYPTO_02401)*

## [SWS_CRYPT_20503]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | ~CryptoObject() |
| **Scope:** | class ara::crypto::cryp::CryptoObject |
| **Syntax:** | `virtual ara::crypto::cryp::CryptoObject::~CryptoObject ()`<br>`noexcept=default;` |
| **Exception Safety:** | noexcept |
| **Header file:** | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| **Description:** | Destructor. |

⌋*(RS_CRYPTO_02005)*

## [SWS_CRYPT_20518]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | Downcast(CryptoObject::Uptrc &&object) | |
| **Scope:** | class ara::crypto::cryp::CryptoObject | |
| **Syntax:** | `template <class ConcreteObject>`<br>`static ara::core::Result<typename ConcreteObject::Uptrc>`<br>`ara::crypto::cryp::CryptoObject::Downcast (CryptoObject::Uptrc`<br>`&&object) noexcept;` | |
| **Template param:** | ConcreteObject | target type (derived from CryptoObject) for downcasting |
| **Parameters (in):** | object | unique smart pointer to the constant generic Crypto Object interface |
| **Return value:** | ara::core::Result< typename Concrete Object::Uptrc > | unique smart pointer to downcasted constant interface of specified derived type |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kBadObjectType | if an actual type of the object is not the specified ConcreteObject |
| **Header file:** | #include "ara/crypto/cryp/cryobj/crypto_object.h" | |
| **Description:** | Downcast and move unique smart pointer from the generic CryptoObject interface to concrete derived object. | |

⌋*(RS_CRYPTO_02005)*

## [SWS_CRYPT_20505]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | GetCryptoPrimitiveId() |
| **Scope:** | class ara::crypto::cryp::CryptoObject |

$\triangledown$

△

| Syntax: | `virtual CryptoPrimitiveId::Uptr ara::crypto::cryp::CryptoObject::Get CryptoPrimitiveId () const noexcept=0;` | |
|---|---|---|
| Return value: | CryptoPrimitiveId::Uptr | – |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" | |
| Description: | Return the CryptoPrimitivId of this CryptoObject. | |

⌋(*RS_CRYPTO_02005*)

## [SWS_CRYPT_20514]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetObjectId() | |
| Scope: | class ara::crypto::cryp::CryptoObject | |
| Syntax: | `virtual COIdentifier ara::crypto::cryp::CryptoObject::GetObjectId () const noexcept=0;` | |
| Return value: | COIdentifier | the object's COIdentifier including the object's type and COUID (or an empty COUID, if this object is not identifiable). |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" | |
| Description: | Return the object's COIdentifier, which includes the object's type and UID. An object that has no assigned COUID cannot be (securely) serialized / exported or saved to a non-volatile storage. An object should not have a COUID if it is session and non-exportable simultaneously A few related objects of different types can share a single COUID (e.g. private and public keys), but a combination of COUID and object type must be unique always! | |

⌋(*RS_CRYPTO_02005*)

## [SWS_CRYPT_20516]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetPayloadSize() | |
| Scope: | class ara::crypto::cryp::CryptoObject | |
| Syntax: | `virtual std::size_t ara::crypto::cryp::CryptoObject::GetPayloadSize () const noexcept=0;` | |
| Return value: | std::size_t | size in bytes of the object's payload required for its storage |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" | |
| Description: | Return actual size of the object's payload. Returned value always must be less than or equal to the maximum payload size expected for this primitive and object type, it is available via call: My Provider().GetPayloadStorageSize(GetObjectType(), GetPrimitiveId()).Value(); Returned value does not take into account the object's meta-information properties, but their size is fixed and common for all crypto objects independently from their actual type. During an allocation of a TrustedContainer, Crypto Providers (and Key Storage Providers) reserve space for an object's meta-information automatically, according to their implementation details. | |

⌋(*RS_CRYPTO_02309*)

## [SWS_CRYPT_20515]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | HasDependence() |
| Scope: | class ara::crypto::cryp::CryptoObject |
| Syntax: | `virtual COIdentifier ara::crypto::cryp::CryptoObject::HasDependence () const noexcept=0;` |
| Return value: | COIdentifier | target COIdentifier of the existing dependence or CryptoObjectType::kUnknown and empty COUID, if the current object does not depend on another CryptoObject |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | Return the COIdentifier of the CryptoObject that this CryptoObject depends on. For signatures objects this method must return a reference to correspondent signature verification public key! Unambiguous identification of a CryptoObject requires both components: CryptoObjectUid and CryptoObjectType. |

⌋(*RS_CRYPTO_02005*)

## [SWS_CRYPT_20513]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IsExportable() |
| Scope: | class ara::crypto::cryp::CryptoObject |
| Syntax: | `virtual bool ara::crypto::cryp::CryptoObject::IsExportable () const noexcept=0;` |
| Return value: | bool | true if the object is exportable (i.e. if it can be exported outside the trusted environment of the Crypto Provider) |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | Get the exportability attribute of the crypto object. An exportable object must have an assigned COUID (see GetObjectId()). |

⌋(*RS_CRYPTO_02005*)

## [SWS_CRYPT_20512]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IsSession() |
| Scope: | class ara::crypto::cryp::CryptoObject |
| Syntax: | `virtual bool ara::crypto::cryp::CryptoObject::IsSession () const noexcept=0;` |
| Return value: | bool | true if the object is temporay (i.e. its life time is limited by the current session only) |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |

▽

△

| Description: | Return the "session" (or "temporary") attribute of the object. A temporary object cannot be saved to a persistent storage location pointed to by an IOInterface! A temporary object will be securely destroyed together with this interface instance! A non-session object must have an assigned COUID (see GetObjectId()). |
|---|---|

*(RS_CRYPTO_02003)*

## [SWS_CRYPT_20517]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Save(IOInterface &container) | |
| Scope: | class ara::crypto::cryp::CryptoObject | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::CryptoObject::Save (IOInterface &container) const noexcept=0;` | |
| Parameters (in): | container | IOInterface representing underlying storage |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the object is "session", but the IOInterface represents a KeySlot. |
| | CryptoErrorDomain::kContent Restrictions | if the object doesn't satisfy the slot restrictions ( |
| | CryptoErrorDomain::kInsufficient Capacity | if the capacity of the target container is not enough, i.e. if (container.Capacity() < this->StorageSize()) |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kUnreserved Resource | if the IOInterface is not opened writeable. |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" | |
| Description: | Save itself to provided IOInterface A CryptoObject with property "session" cannot be saved in a KeySlot. | |

*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30208]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const CryptoObject &other) | |
| Scope: | class ara::crypto::cryp::CryptoObject | |
| Syntax: | `CryptoObject& ara::crypto::cryp::CryptoObject::operator= (const CryptoObject &other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoObject & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" | |
| Description: | Copy-assign another CryptoObject to this instance. | |

*(RS_CRYPTO_02009)*

**[SWS_CRYPT_30209]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(CryptoObject &&other) | |
| Scope: | class ara::crypto::cryp::CryptoObject | |
| Syntax: | CryptoObject& ara::crypto::cryp::CryptoObject::operator= (CryptoObject &&other)=default; | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoObject & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" | |
| Description: | Move-assign another CryptoObject to this instance. | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_10808]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~CryptoPrimitiveId() |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId |
| Syntax: | virtual ara::crypto::cryp::CryptoPrimitiveId::~CryptoPrimitiveId () noexcept=default; |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_20652]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetPrimitiveId() | |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId | |
| Syntax: | virtual AlgId ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveId () const noexcept=0; | |
| Return value: | AlgId | the binary Crypto Primitive ID |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" | |
| Description: | Get vendor specific ID of the primitive. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_20651]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetPrimitiveName() | |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId | |
| Syntax: | virtual const ara::core::StringView ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveName () const noexcept=0; | |
| Return value: | const ara::core::StringView | the unified name of the crypto primitive |

▽

△

| Exception Safety: | noexcept |
|---|---|
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" |
| Description: | Get a unified name of the primitive. The crypto primitive name can be fully or partially specified (see "Crypto Primitives Naming Convention" for more details). The life-time of the returned StringView instance should not exceed the life-time of this CryptoPrimitiveId instance! |

⌋*(RS_CRYPTO_02308)*

## [SWS_CRYPT_30212]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const CryptoPrimitiveId &other) | |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId | |
| Syntax: | `CryptoPrimitiveId& ara::crypto::cryp::CryptoPrimitiveId::operator= (const CryptoPrimitiveId &other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoPrimitiveId & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" | |
| Description: | Copy-assign another CryptoPrimitiveId to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30213]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(CryptoPrimitiveId &&other) | |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId | |
| Syntax: | `CryptoPrimitiveId& ara::crypto::cryp::CryptoPrimitiveId::operator= ( CryptoPrimitiveId &&other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoPrimitiveId & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" | |
| Description: | Move-assign another CryptoPrimitiveId to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_20726]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | AllocVolatileContainer(std::size_t capacity=0) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<VolatileTrustedContainer::Uptr> ara::crypto::cryp::CryptoProvider::AllocVolatileContainer (std::size_t capacity=0) noexcept=0;` | |
| Parameters (in): | capacity | the capacity required for this volatile trusted container (in bytes) |
| Return value: | ara::core::Result< VolatileTrusted Container::Uptr > | unique smart pointer to an allocated volatile trusted container |
| Exception Safety: | noexcept | |

▽

△

| Thread Safety: | Thread-safe |
|---|---|
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" |
| Description: | Allocate a Volatile (virtual) Trusted Container according to directly specified capacity. The Volatile Trusted Container can be used for execution of the import operations. Current process obtains the "Owner" rights for allocated Container. If (capacity == 0) then the capacity of the container will be selected automatically according to a maximal size of supported crypto objects. A few volatile (temporary) containers can coexist at same time without any affecting each-other. |

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

**[SWS_CRYPT_20727]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | AllocVolatileContainer(std::pair< AlgId, CryptoObjectType > theObjectDef) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<VolatileTrustedContainer::Uptr>`<br>`ara::crypto::cryp::CryptoProvider::AllocVolatileContainer (std::pair<`<br>`AlgId, CryptoObjectType > theObjectDef) noexcept=0;` | |
| Parameters (in): | theObjectDef | the list of objects that can be stored to this volatile trusted container |
| Return value: | ara::core::Result< VolatileTrusted Container::Uptr > | unique smart pointer to an allocated volatile trusted container |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if unsupported combination of object type and algorithm ID presents in the list |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Allocate a Volatile (virtual) Trusted Container according to indirect specification of a minimal required capacity for hosting of any listed object. The Volatile Trusted Container can be used for execution of the import operations. Current process obtains the "Owner" rights for allocated Container. Real container capacity is calculated as a maximal storage size of all listed objects. | |

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

**[SWS_CRYPT_20711]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ConvertToAlgId(ara::core::StringView primitiveName) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual AlgId ara::crypto::cryp::CryptoProvider::ConvertToAlgId (`<br>`ara::core::StringView primitiveName) const noexcept=0;` | |
| Parameters (in): | primitiveName | the unified name of the crypto primitive (see "Crypto Primitives Naming Convention" for more details) |
| Return value: | AlgId | vendor specific binary algorithm ID or kAlgId Undefined if a primitive with provided name is not supported |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Convert a common name of crypto algorithm to a correspondent vendor specific binary algorithm ID. | |

⌋*(RS_CRYPTO_02308)*

### [SWS_CRYPT_20712]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ConvertToAlgName(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<ara::core::String> ara::crypto::cryp::Crypto Provider::ConvertToAlgName (AlgId algId) const noexcept=0;` | |
| Parameters (in): | algId | the vendor specific binary algorithm ID |
| Return value: | ara::core::Result< ara::core::String > | the common name of the crypto algorithm (see "Crypto Primitives Naming Convention" for more details) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Convert a vendor specific binary algorithm ID to a correspondent common name of the crypto algorithm. | |

⌋*(RS_CRYPTO_02308)*

### [SWS_CRYPT_20745]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateAuthCipherCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<AuthCipherCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateAuthCipherCtx (AlgId algId) noexcept=0;` | |
| Parameters (in): | algId | identifier of the target crypto algorithm |
| Return value: | ara::core::Result< AuthCipherCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from symmetric authenticated stream cipher |
| | CryptoErrorDomain::kInvalidArgument | – |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a symmetric authenticated cipher context. | |

⌋*(RS_CRYPTO_02207, RS_AP_00144)*

### [SWS_CRYPT_20751]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CreateDecryptorPrivateCtx(AlgId algId) |
| Scope: | class ara::crypto::cryp::CryptoProvider |

▽

△

| Syntax: | `virtual ara::core::Result<DecryptorPrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateDecryptorPrivateCtx (AlgId algId) noexcept=0;` | |
|---|---|---|
| Parameters (in): | algId | identifier of the target asymmetric encryption/ decryption algorithm |
| Return value: | ara::core::Result< DecryptorPrivate Ctx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a decryption private key context. | |

⌊*(RS_CRYPTO_02202, RS_AP_00144)*

## [SWS_CRYPT_20750]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateEncryptorPublicCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<EncryptorPublicCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateEncryptorPublicCtx (AlgId algId) noexcept=0;` | |
| Parameters (in): | algId | identifier of the target asymmetric encryption/ decryption algorithm |
| Return value: | ara::core::Result< EncryptorPublic Ctx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create an encryption public key context. | |

⌊*(RS_CRYPTO_02202, RS_AP_00144)*

## [SWS_CRYPT_20761]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateHashDigest(AlgId hashAlgId, ReadOnlyMemRegion value) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::Crypto Provider::CreateHashDigest (AlgId hashAlgId, ReadOnlyMemRegion value) noexcept=0;` | |
| Parameters (in): | hashAlgId | identifier of an applied hash function crypto algorithm |

▽

△

| | value | raw BLOB value of the hash digest |
|---|---|---|
| *Return value:* | ara::core::Result< Signature::Uptrc > | unique smart pointer to the created Signature object |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Errors:* | CryptoErrorDomain::kUnknown Identifier | if hashAlgId argument has unsupported value |
| | CryptoErrorDomain::kInvalidArgument | if hashAlgId argument specifies crypto algorithm different from a hash function |
| | CryptoErrorDomain::kInvalidInputSize | if the value argument has invalid size (i.e. incompatible with the hashAlgId argument) |
| *Header file:* | #include "ara/crypto/cryp/crypto_provider.h" | |
| *Description:* | Construct Signature object from directly provided components of a hash digest. | |

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006, RS_AP_00144)*

**[SWS_CRYPT_20747]**{DRAFT} ⌈

| *Kind:* | function | |
|---|---|---|
| *Symbol:* | CreateHashFunctionCtx(AlgId algId) | |
| *Scope:* | class ara::crypto::cryp::CryptoProvider | |
| *Syntax:* | `virtual ara::core::Result<HashFunctionCtx::Uptr>`<br>`ara::crypto::cryp::CryptoProvider::CreateHashFunctionCtx (AlgId algId)`<br>`noexcept=0;` | |
| *Parameters (in):* | algId | identifier of the target crypto algorithm |
| *Return value:* | ara::core::Result< HashFunction Ctx::Uptr > | unique smart pointer to the created context |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Errors:* | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from hash function |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| *Header file:* | #include "ara/crypto/cryp/crypto_provider.h" | |
| *Description:* | Create a hash function context. | |

⌋*(RS_CRYPTO_02205, RS_AP_00144)*

**[SWS_CRYPT_20758]**{DRAFT} ⌈

| *Kind:* | function | |
|---|---|---|
| *Symbol:* | CreateKeyAgreementPrivateCtx(AlgId algId) | |
| *Scope:* | class ara::crypto::cryp::CryptoProvider | |
| *Syntax:* | `virtual ara::core::Result<KeyAgreementPrivateCtx::Uptr>`<br>`ara::crypto::cryp::CryptoProvider::CreateKeyAgreementPrivateCtx (AlgId`<br>`algId) noexcept=0;` | |
| *Parameters (in):* | algId | identifier of the target key-agreement crypto algorithm |
| *Return value:* | ara::core::Result< KeyAgreement PrivateCtx::Uptr > | unique smart pointer to the created context |
| *Exception Safety:* | noexcept | |

▽

△

| Thread Safety: | Thread-safe | |
|---|---|---|
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from key-agreement |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a key-agreement private key context. | |

⌋(*RS_CRYPTO_02104*, *RS_AP_00144*)

## [SWS_CRYPT_20753]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateKeyDecapsulatorPrivateCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | virtual ara::core::Result<KeyDecapsulatorPrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateKeyDecapsulatorPrivateCtx ( AlgId algId) noexcept=0; | |
| Parameters (in): | algId | identifier of the target KEM crypto algorithm |
| Return value: | ara::core::Result< KeyDecapsulator PrivateCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from asymmetric KEM |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a key-decapsulator private key context of a Key Encapsulation Mechanism (KEM). | |

⌋(*RS_CRYPTO_02104*, *RS_CRYPTO_02209*, *RS_AP_00144*)

## [SWS_CRYPT_20748]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateKeyDerivationFunctionCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | virtual ara::core::Result<KeyDerivationFunctionCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateKeyDerivationFunctionCtx (Alg Id algId) noexcept=0; | |
| Parameters (in): | algId | identifier of the target crypto algorithm |
| Return value: | ara::core::Result< KeyDerivation FunctionCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from key derivation function |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a key derivation function context. | |

⌋(*RS_CRYPTO_02103*, *RS_AP_00144*)

### [SWS_CRYPT_20752]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CreateKeyEncapsulatorPublicCtx(AlgId algId) |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Syntax: | `virtual ara::core::Result<KeyEncapsulatorPublicCtx::Uptr>`<br>`ara::crypto::cryp::CryptoProvider::CreateKeyEncapsulatorPublicCtx (AlgId algId) noexcept=0;` |
| Parameters (in): | algId | identifier of the target KEM crypto algorithm |
| Return value: | ara::core::Result< KeyEncapsulator PublicCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from asymmetric KEM |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a key-encapsulator public key context of a Key Encapsulation Mechanism (KEM). | |

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02209, RS_AP_00144)*

### [SWS_CRYPT_20746]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CreateMessageAuthnCodeCtx(AlgId algId) |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Syntax: | `virtual ara::core::Result<MessageAuthnCodeCtx::Uptr>`<br>`ara::crypto::cryp::CryptoProvider::CreateMessageAuthnCodeCtx (AlgId algId) noexcept=0;` |
| Parameters (in): | algId | identifier of the target crypto algorithm |
| Return value: | ara::core::Result< MessageAuthnCode Ctx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from symmetric message authentication code |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a symmetric message authentication code context. | |

⌋*(RS_CRYPTO_02203, RS_AP_00144)*

### [SWS_CRYPT_20755]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CreateMsgRecoveryPublicCtx(AlgId algId) |
| Scope: | class ara::crypto::cryp::CryptoProvider |

▽

△

| Syntax: | `virtual ara::core::Result<MsgRecoveryPublicCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateMsgRecoveryPublicCtx (AlgId algId) noexcept=0;` | |
|---|---|---|
| Parameters (in): | algId | identifier of the target asymmetric crypto algorithm |
| Return value: | ara::core::Result< MsgRecoveryPublic Ctx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a message recovery public key context. | |

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204, RS_AP_00144)*

## [SWS_CRYPT_20741]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateRandomGeneratorCtx(AlgId algId=kAlgIdDefault, bool initialize=true) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<RandomGeneratorCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx (AlgId algId=kAlgIdDefault, bool initialize=true) noexcept=0;` | |
| Parameters (in): | algId | identifier of target RNG algorithm. If no algId is given, the default RNG is returned |
| | initialize | indicates whether the returned context shall be initialized (i.e., seeded) by the stack |
| Return value: | ara::core::Result< RandomGenerator Ctx::Uptr > | unique smart pointer to the created RNG context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value or if (alg Id == kAlgIdDefault) and the CryptoProvider does not provide any RandomGeneratorCtx |
| | CryptoErrorDomain::kBusyResource | if (initialize == true) but the context currently cannot be seeded (e.g., due to a lack of entropy) |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a Random Number Generator (RNG) context. | |

⌋*(RS_CRYPTO_02206)*

## [SWS_CRYPT_20754]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CreateSigEncodePrivateCtx(AlgId algId) |
| Scope: | class ara::crypto::cryp::CryptoProvider |

▽

△

| Syntax: | virtual ara::core::Result<SigEncodePrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSigEncodePrivateCtx (AlgId algId) noexcept=0; | |
|---|---|---|
| Parameters (in): | algId | identifier of the target asymmetric crypto algorithm |
| Return value: | ara::core::Result< SigEncodePrivate Ctx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a signature encoding private key context. | |

⌋(*RS_CRYPTO_02202*, *RS_CRYPTO_02204*, *RS_AP_00144*)

## [SWS_CRYPT_20760]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateSignature(AlgId signAlgId, ReadOnlyMemRegion value, const RestrictedUseObject &key, AlgId hashAlgId=kAlgIdNone) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::Crypto Provider::CreateSignature (AlgId signAlgId, ReadOnlyMemRegion value, const RestrictedUseObject &key, AlgId hashAlgId=kAlgIdNone) noexcept=0; | |
| Parameters (in): | signAlgId | identifier of an applied signature/MAC/AE/AEAD crypto algorithm |
| | value | raw BLOB value of the signature/MAC |
| | key | symmetric or asymmetric key (according to signAlg Id) applied for the sign or MAC/AE/AEAD operation |
| | hashAlgId | identifier of a hash function algorithm applied together with the signature algorithm |
| Return value: | ara::core::Result< Signature::Uptrc > | unique smart pointer to the created Signature object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if signAlgId or hashAlgId arguments have unsupported values |
| | CryptoErrorDomain::kInvalidArgument | if signAlgId or hashAlgId arguments specify crypto algorithms different from the signature/MAC/AE/ AEAD and message digest respectively |
| | CryptoErrorDomain::kIncompatible Arguments | if signAlgId and hashAlgId arguments specify incompatible algorithms (if signAlgId includes hash function specification) or if a crypto primitive associated with the key argument is incompatible with provided signAlgId or hashAlgId arguments |
| | CryptoErrorDomain::kInvalidInputSize | if the value argument has invalid size (i.e. incompatible with the signAlgId argument) |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |

▽

△

| Description: | Construct Signature object from directly provided components of a digital signature/MAC or authenticated encryption (AE/AEAD). All integers inside a digital signature BLOB value are always presented in Big Endian bytes order (i.e. MSF - Most Significant byte First). |
|---|---|

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006, RS_AP_00144)*

## [SWS_CRYPT_20756]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateSignerPrivateCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<SignerPrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSignerPrivateCtx (AlgId algId) noexcept=0;` | |
| Parameters (in): | algId | identifier of the target signature crypto algorithm |
| Return value: | ara::core::Result< SignerPrivateCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from private key signature |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a signature private key context. | |

⌋*(RS_CRYPTO_02204, RS_AP_00144)*

## [SWS_CRYPT_20744]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateStreamCipherCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<StreamCipherCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateStreamCipherCtx (AlgId algId) noexcept=0;` | |
| Parameters (in): | algId | identifier of the target crypto algorithm |
| Return value: | ara::core::Result< StreamCipherCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from symmetric stream cipher |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a symmetric stream cipher context. | |

⌋*(RS_CRYPTO_02201)*

### [SWS_CRYPT_20742]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateSymmetricBlockCipherCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<SymmetricBlockCipherCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSymmetricBlockCipherCtx (AlgId algId) noexcept=0;` | |
| Parameters (in): | algId | identifier of the target crypto algorithm |
| Return value: | ara::core::Result< SymmetricBlock CipherCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a symmetric block cipher context. | |

⌋*(RS_CRYPTO_02201)*

### [SWS_CRYPT_20743]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateSymmetricKeyWrapperCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<SymmetricKeyWrapperCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSymmetricKeyWrapperCtx (AlgId algId) noexcept=0;` | |
| Parameters (in): | algId | identifier of the target crypto algorithm |
| Return value: | ara::core::Result< SymmetricKey WrapperCtx::Uptr > | unique smart pointer to the created context |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from symmetric key-wrapping |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a symmetric key-wrap algorithm context. | |

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02208)*

### [SWS_CRYPT_20757]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateVerifierPublicCtx(AlgId algId) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<VerifierPublicCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateVerifierPublicCtx (AlgId algId) noexcept=0;` | |
| Parameters (in): | algId | identifier of the target signature crypto algorithm |

▽

△

| Return value: | ara::core::Result< VerifierPublic Ctx::Uptr > | unique smart pointer to the created context |
|---|---|---|
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if algId argument specifies a crypto algorithm different from public key signature verification |
| | CryptoErrorDomain::kUnknown Identifier | if algId argument has an unsupported value |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Create a signature verification public key context. | |

⌋*(RS_CRYPTO_02204, RS_AP_00144)*

**[SWS_CRYPT_20710]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~CryptoProvider() |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Syntax: | `virtual ara::crypto::cryp::CryptoProvider::~CryptoProvider () noexcept=default;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02107)*

**[SWS_CRYPT_20731]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ExportPublicObject(const IOInterface &container, Serializable::FormatId format Id=Serializable::kFormatDefault) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::CryptoProvider::ExportPublicObject (const IOInterface &container, Serializable::FormatId format Id=Serializable::kFormatDefault) noexcept=0;` | |
| Parameters (in): | container | the IOInterface that contains an object for export |
| | formatId | the CryptoProvider specific identifier of the output format |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | actual capacity required for the serialized data |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kEmptyContainer | if the container is empty |
| | CryptoErrorDomain::kUnexpected Value | if the container contains a secret crypto object |
| | CryptoErrorDomain::kInsufficient Capacity | if (serialized.empty() == false), but its capacity is not enough for storing result |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |

▽

△

| Header file: | #include "ara/crypto/cryp/crypto_provider.h" |
|---|---|
| Description: | Export publicly an object from a IOInterface (i.e. without an intermediate creation of a crypto object). |

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

## [SWS_CRYPT_20728]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ExportSecuredObject(const CryptoObject &object, SymmetricKeyWrapperCtx &transport Context) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::CryptoProvider::ExportSecuredObject (const Crypto Object &object, SymmetricKeyWrapperCtx &transportContext) noexcept=0;` | |
| Parameters (in): | object | the crypto object for export |
| | transportContext | the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyExporting) |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | the wrapped crypto object data |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the object cannot be exported due to Is Exportable() returning flase |
| | CryptoErrorDomain::kIncompleteArg State | if the transportContext is not initialized |
| | CryptoErrorDomain::kIncompatible Object | if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method) |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Export a crypto object in a secure manner. if (serialized.empty() == true) then the method returns required size only, but content of the transportContext stays unchanged! Only an exportable and completed object (i.e. that have a UUID) can be exported! | |

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

## [SWS_CRYPT_20729]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ExportSecuredObject(const IOInterface &container, SymmetricKeyWrapperCtx &transport Context) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::CryptoProvider::ExportSecuredObject (const IOInterface &container, SymmetricKeyWrapperCtx &transportContext) noexcept=0;` | |
| Parameters (in): | container | the IOInterface that refers an object for export |
| | transportContext | the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyExporting) |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | actual capacity required for the serialized data |
| Exception Safety: | noexcept | |

▽

△

| Thread Safety: | Thread-safe | |
|---|---|---|
| Errors: | CryptoErrorDomain::kEmptyContainer | if the container is empty |
| | CryptoErrorDomain::kInsufficient Capacity | if size of the serialized buffer is not enough for saving the output data |
| | CryptoErrorDomain::kIncompleteArg State | if the transportContext is not initialized |
| | CryptoErrorDomain::kIncompatible Object | if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method) |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Export securely an object directly from an IOInterface (i.e. without an intermediate creation of a crypto object). if (serialized == nullptr) then the method returns required size only, but content of the transportContext stays unchanged. This method can be used for re-exporting of just imported object but on another transport key. | |

⌋(RS_CRYPTO_02105, RS_CRYPTO_02112)

## [SWS_CRYPT_20722]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GeneratePrivateKey(AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=false, bool isExportable=false) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<PrivateKey::Uptrc> ara::crypto::cryp::CryptoProvider::GeneratePrivateKey (AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=false, bool isExportable=false) noexcept=0;` | |
| Parameters (in): | algId | the identifier of target public-private key crypto algorithm |
| | allowedUsage | the flags that define a list of allowed transformations' types in which the target key can be used (see constants in scope of RestrictedUseObject) |
| | isSession | the "session" (or "temporary") attribute for the target key (if true) |
| | isExportable | the exportability attribute of the target key (if true) |
| Return value: | ara::core::Result< PrivateKey::Uptrc > | smart unique pointer to the created private key object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if algId has an unsupported value |
| | CryptoErrorDomain::kIncompatible Arguments | if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method) |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Allocate a new private key context of correspondent type and generates the key value randomly. A common COUID should be shared for both private and public keys. Any serializable (i.e. savable/non-session or exportable) key must generate own COUID! | |

⌋(RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_CRYPTO_-02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02115)

**[SWS_CRYPT_20723]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GenerateSeed(AlgId algId, SecretSeed::Usage allowedUsage, bool isSession=true, bool isExportable=false) |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Syntax: | `virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::CryptoProvider::GenerateSeed (AlgId algId, SecretSeed::Usage allowedUsage, bool isSession=true, bool isExportable=false) noexcept=0;` |
| Parameters (in): | algId | the identifier of target crypto algorithm |
| | allowedUsage | the lags that define a list of allowed transformations' types in which the target seed can be used (see constants in scope of RestrictedUseObject) |
| | isSession | the "session" (or "temporary") attribute of the target seed (if true) |
| | isExportable | the exportability attribute of the target seed (if true) |
| Return value: | ara::core::Result< SecretSeed::Uptrc > | unique smart pointer to generated SecretSeed object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknownIdentifier | if algId has an unsupported value |
| | CryptoErrorDomain::kIncompatibleArguments | if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method) |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Generate a random Secret Seed object of requested algorithm. | |

⌋*(RS_CRYPTO_02007)*

**[SWS_CRYPT_20721]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GenerateSymmetricKey(AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=true, bool isExportable=false) |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Syntax: | `virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::CryptoProvider::GenerateSymmetricKey (AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=true, bool isExportable=false) noexcept=0;` |
| Parameters (in): | algId | the identifier of target symmetric crypto algorithm |
| | allowedUsage | the flags that define a list of allowed transformations' types in which the target key can be used (see constants in scope of RestrictedUseObject) |
| | isSession | the "session" (or "temporary") attribute of the target key (if true) |
| | isExportable | the exportability attribute of the target key (if true) |
| Return value: | ara::core::Result< SymmetricKey::Uptrc > | smart unique pointer to the created symmetric key object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknownIdentifier | if algId has an unsupported value |

▽

△

| | CryptoErrorDomain::kIncompatible Arguments | if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method) |
|---|---|---|
| **Header file:** | #include "ara/crypto/cryp/crypto_provider.h" | |
| **Description:** | Allocate a new symmetric key object and fill it by a new randomly generated value. Any serializable (i.e. savable/non-session or exportable) key must generate own COUID! By default Crypto Provider should use an internal instance of a best from all supported RNG (ideally TRNG). | |

⌋*(RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_CRYPTO_-02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02115)*

## [SWS_CRYPT_20725]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | GetPayloadStorageSize(CryptoObjectType cryptoObjectType, AlgId algId) | |
| **Scope:** | class ara::crypto::cryp::CryptoProvider | |
| **Syntax:** | `virtual ara::core::Result<std::size_t> ara::crypto::cryp::CryptoProvider::GetPayloadStorageSize (CryptoObjectType cryptoObjectType, AlgId algId) const noexcept=0;` | |
| **Parameters (in):** | cryptoObjectType | the type of the target object |
| | algId | a CryptoProvider algorithm ID of the target object |
| **Return value:** | ara::core::Result< std::size_t > | minimal size required for storing of the object in a TrustedContainer (persistent or volatile) |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUnknown Identifier | if any argument has an unsupported value |
| | CryptoErrorDomain::kIncompatible Arguments | if the arguments are incompatible |
| **Header file:** | #include "ara/crypto/cryp/crypto_provider.h" | |
| **Description:** | Return minimally required capacity of a key slot for saving of the object's payload. Returned value does not take into account the object's meta-information properties, but their size is fixed and common for all crypto objects independently from their actual type. During an allocation of a TrustedContainer, Crypto Providers (and Key Storage Providers) reserve space for an object's meta-information automatically, according to their implementation details. | |

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

## [SWS_CRYPT_20724]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | GetSerializedSize(CryptoObjectType cryptoObjectType, AlgId algId, Serializable::FormatId formatId=Serializable::kFormatDefault) | |
| **Scope:** | class ara::crypto::cryp::CryptoProvider | |
| **Syntax:** | `virtual ara::core::Result<std::size_t> ara::crypto::cryp::CryptoProvider::GetSerializedSize (CryptoObjectType cryptoObjectType, AlgId algId, Serializable::FormatId formatId=Serializable::kFormatDefault) const noexcept=0;` | |
| **Parameters (in):** | cryptoObjectType | the type of the target object |
| | algId | the Crypto Provider algorithm ID of the target object |

▽

△

| | formatId | the Crypto Provider specific identifier of the output format |
|---|---|---|
| **Return value:** | ara::core::Result< std::size_t > | size required for storing of the object serialized in the specified format |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUnknown Identifier | if any argument has an unsupported value |
| | CryptoErrorDomain::kIncompatible Arguments | if any pair of the arguments are incompatible |
| **Header file:** | #include "ara/crypto/cryp/crypto_provider.h" | |
| **Description:** | Return required buffer size for serialization of an object in specific format. | |

⌋(*RS_CRYPTO_02005*, *RS_CRYPTO_02006*)

## **[SWS_CRYPT_20732]**{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | ImportPublicObject(IOInterface &container, ReadOnlyMemRegion serialized, CryptoObject Type expectedObject=CryptoObjectType::kUndefined) | |
| **Scope:** | class ara::crypto::cryp::CryptoProvider | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::Crypto Provider::ImportPublicObject (IOInterface &container, ReadOnlyMem Region serialized, CryptoObjectType expectedObject=CryptoObjectType::k Undefined) noexcept=0;` | |
| **Parameters (in):** | serialized | the memory region that contains a securely serialized object that should be imported to the IOInterface |
| | expectedObject | the expected object type (default value CryptoObject Type::kUnknown means without check) |
| **Parameters (out):** | container | the IOInterface for storing of the imported object |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUnexpected Value | if the serialized contains incorrect data |
| | CryptoErrorDomain::kBadObjectType | if (expectedObject != CryptoObjectType::k Unknown), but the actual object type differs from the expected one |
| | CryptoErrorDomain::kInsufficient Capacity | if capacity of the container is not enough to save the de-serialized object |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kUnreserved Resource | if the IOInterface is not opened writable. |
| **Header file:** | #include "ara/crypto/cryp/crypto_provider.h" | |
| **Description:** | Import publicly serialized object to a storage location pointed to by an IOInterface for following processing (without allocation of a crypto object). If (expectedObject != CryptoObjectType::k Unknown) and an actual object type differs from the expected one then this method fails. If the serialized contains incorrect data then this method fails. | |

⌋(*RS_CRYPTO_02105*, *RS_CRYPTO_02112*)

**[SWS_CRYPT_20730]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ImportSecuredObject(IOInterface &container, ReadOnlyMemRegion serialized, SymmetricKey WrapperCtx &transportContext, bool isExportable=false, CryptoObjectType expected Object=CryptoObjectType::kUndefined) |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::Crypto Provider::ImportSecuredObject (IOInterface &container, ReadOnlyMem Region serialized, SymmetricKeyWrapperCtx &transportContext, bool is Exportable=false, CryptoObjectType expectedObject=CryptoObjectType::k Undefined) noexcept=0;` |

| Parameters (in): | serialized | the memory region that contains a securely serialized object that should be imported to the IOInterface |
|---|---|---|
| | transportContext | the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyImporting) |
| | isExportable | the exportability attribute of the target object |
| | expectedObject | the expected object type (default value CryptoObject Type::kUnknown means without check) |
| Parameters (out): | container | the IOInterface for storing of the imported object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnexpected Value | if the serialized contains incorrect data |
| | CryptoErrorDomain::kBadObjectType | if (expectedObject != CryptoObjectType::k Unknown), but the actual object type differs from the expected one |
| | CryptoErrorDomain::kIncompleteArg State | if the transportContext is not initialized |
| | CryptoErrorDomain::kIncompatible Object | if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method) |
| | CryptoErrorDomain::kInsufficient Capacity | if capacity of the container is not enough to save the deserialized object |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kUnreserved Resource | if the IOInterface is not opened writeable. |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" |
| Description: | Import securely serialized object to the persistent or volatile storage represented by an IOInterface for following processing. |

⌋(*RS_CRYPTO_02105*, *RS_CRYPTO_02112*)

**[SWS_CRYPT_20733]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | LoadObject(const IOInterface &container) |
| Scope: | class ara::crypto::cryp::CryptoProvider |

▽

$\triangle$

| Syntax: | `virtual ara::core::Result<CryptoObject::Uptrc> ara::crypto::cryp::CryptoProvider::LoadObject (const IOInterface &container) noexcept=0;` | |
|---|---|---|
| Parameters (in): | container | the IOInterface that contains the crypto object for loading |
| Return value: | ara::core::Result< CryptoObject::Uptrc > | unique smart pointer to the created object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kEmptyContainer | if the container is empty |
| | CryptoErrorDomain::kResourceFault | if the container content is damaged |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kIncompatible Object | if the underlying resource belongs to another, incompatible CryptoProvider |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Load any crypto object from the IOInterface provided. | |
| Notes: | This method is one of the "binding" methods between a CryptoProvider and the Key Storage Provider. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02002*, *RS_CRYPTO_02005*, *RS_CRYPTO_-02006*)

**[SWS_CRYPT_20764]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | LoadPrivateKey(const IOInterface &container) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<PrivateKey::Uptrc> ara::crypto::cryp::CryptoProvider::LoadPrivateKey (const IOInterface &container) noexcept=0;` | |
| Parameters (in): | container | the IOInterface that contains the crypto object for loading |
| Return value: | ara::core::Result< PrivateKey::Uptrc > | unique smart pointer to the PrivateKey |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kEmptyContainer | if the container is empty |
| | CryptoErrorDomain::kResourceFault | if the container content is damaged |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kIncompatible Object | if the underlying resource belongs to another, incompatible CryptoProvider |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Load a private key from the IOInterface provided. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02002*)

**[SWS_CRYPT_20763]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | LoadPublicKey(const IOInterface &container) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<PublicKey::Uptrc> ara::crypto::cryp::Crypto Provider::LoadPublicKey (const IOInterface &container) noexcept=0;` | |
| Parameters (in): | container | the IOInterface that contains the crypto object for loading |
| Return value: | ara::core::Result< PublicKey::Uptrc > | unique smart pointer to the PublicKey |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kEmptyContainer | if the container is empty |
| | CryptoErrorDomain::kResourceFault | if the container content is damaged |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kIncompatible Object | if the underlying resource belongs to another, incompatible CryptoProvider |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Load a public key from the IOInterface provided. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002)*

**[SWS_CRYPT_20765]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | LoadSecretSeed(const IOInterface &container) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | `virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::Crypto Provider::LoadSecretSeed (const IOInterface &container) noexcept=0;` | |
| Parameters (in): | container | the IOInterface that contains the crypto object for loading |
| Return value: | ara::core::Result< SecretSeed::Uptrc > | unique smart pointer to the SecretSeed |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kEmptyContainer | if the container is empty |
| | CryptoErrorDomain::kResourceFault | if the container content is damaged |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kIncompatible Object | if the underlying resource belongs to another, incompatible CryptoProvider |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Load secret seed from the IOInterface provided. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002)*

**[SWS_CRYPT_20762]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | LoadSymmetricKey(const IOInterface &container) |
| Scope: | class ara::crypto::cryp::CryptoProvider |

▽

△

| Syntax: | virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::CryptoProvider::LoadSymmetricKey (const IOInterface &container) noexcept=0; | |
|---|---|---|
| Parameters (in): | container | the IOInterface that contains the crypto object for loading |
| Return value: | ara::core::Result< Symmetric Key::Uptrc > | unique smart pointer to the SymmetricKey |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kEmptyContainer | if the container is empty |
| | CryptoErrorDomain::kResourceFault | if the container content is damaged |
| | CryptoErrorDomain::kModified Resource | if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated. |
| | CryptoErrorDomain::kIncompatible Object | if the underlying resource belongs to another, incompatible CryptoProvider |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Load a symmetric key from the IOInterface provided. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002)*

## [SWS_CRYPT_29023]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetBlockSize() |
| Scope: | class ara::crypto::cryp::CryptoService |
| Syntax: | virtual std::size_t ara::crypto::cryp::CryptoService::GetBlockSize () const noexcept=0; |
| Return value: | std::size_t | size of the block in bytes |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/crypto_service.h" |
| Description: | Get block (or internal buffer) size of the base algorithm. For digest, byte-wise stream cipher and RNG contexts it is an informative method, intended only for optimization of the interface usage. |

⌋*(RS_CRYPTO_02309)*

## [SWS_CRYPT_29021]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetMaxInputSize(bool suppressPadding=false) | |
| Scope: | class ara::crypto::cryp::CryptoService | |
| Syntax: | virtual std::size_t ara::crypto::cryp::CryptoService::GetMaxInputSize (bool suppressPadding=false) const noexcept=0; | |
| Parameters (in): | suppressPadding | if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only |
| Return value: | std::size_t | maximum size of the input data block in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/crypto_service.h" | |
| Description: | Get maximum expected size of the input data block. suppressPadding argument and it will be equal to the block size. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29022]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetMaxOutputSize(bool suppressPadding=false) | |
| Scope: | class ara::crypto::cryp::CryptoService | |
| Syntax: | virtual std::size_t ara::crypto::cryp::CryptoService::GetMaxOutputSize (bool suppressPadding=false) const noexcept=0; | |
| Parameters (in): | suppressPadding | if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only |
| Return value: | std::size_t | maximum size of the output data block in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/crypto_service.h" | |
| Description: | Get maximum possible size of the output data block. If (IsEncryption() == true) then a value returned by this method is independent from the suppressPadding argument and will be equal to the block size. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_30216]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const CryptoProvider &other) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | CryptoProvider& ara::crypto::cryp::CryptoProvider::operator= (const CryptoProvider &other)=default; | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoProvider & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Copy-assign another CryptoProvider to this instance. | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_30217]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(CryptoProvider &&other) | |
| Scope: | class ara::crypto::cryp::CryptoProvider | |
| Syntax: | CryptoProvider& ara::crypto::cryp::CryptoProvider::operator= (Crypto Provider &&other)=default; | |
| Parameters (in): | other | the other instance |
| Return value: | CryptoProvider & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" | |
| Description: | Move-assign another CryptoProvider to this instance. | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_20802]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetCryptoService() |
| Scope: | class ara::crypto::cryp::DecryptorPrivateCtx |
| Syntax: | virtual CryptoService::Uptr ara::crypto::cryp::DecryptorPrivate<br>Ctx::GetCryptoService () const noexcept=0; |
| Return value: | CryptoService::Uptr | – |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/decryptor_private_ctx.h" |
| Description: | Get CryptoService instance. |

⌋*(RS_CRYPTO_02006)*

**[SWS_CRYPT_20812]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false) | |
| Scope: | class ara::crypto::cryp::DecryptorPrivateCtx | |
| Syntax: | virtual ara::core::Result<ara::core::Vector<ara::core::Byte> ><br>ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock (ReadOnlyMem<br>Region in, bool suppressPadding=false) const noexcept=0; | |
| Parameters (in): | in | the input data block |
| | suppressPadding | if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kInsufficient Capacity | if the out.size() is not enough to store the transformation result |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/decryptor_private_ctx.h" | |
| Description: | Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutput Size(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppress Padding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! | |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_20813]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false) | |
| **Scope:** | class ara::crypto::cryp::DecryptorPrivateCtx | |
| **Syntax:** | `template <typename Alloc = <implementation-defined>>`<br>`ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::Decryptor`<br>`PrivateCtx::ProcessBlock (ReadOnlyMemRegion in, bool suppress`<br>`Padding=false) const noexcept;` | |
| **Template param:** | Alloc | a custom allocator type of the output container |
| **Parameters (in):** | in | the input data block |
| | suppressPadding | if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data |
| **Return value:** | ara::core::Result< ByteVector< Alloc > > | the managed container for output block |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kInsufficient Capacity | if the out.size() is not enough to store the transformation result |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| **Header file:** | #include "ara/crypto/cryp/decryptor_private_ctx.h" | |
| **Description:** | Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! | |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_20811]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | Reset() | |
| **Scope:** | class ara::crypto::cryp::DecryptorPrivateCtx | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::DecryptorPrivate`<br>`Ctx::Reset () noexcept=0;` | |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/decryptor_private_ctx.h" | |
| **Description:** | Clear the crypto context. | |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_20810]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetKey(const PrivateKey &key) |
| Scope: | class ara::crypto::cryp::DecryptorPrivateCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::DecryptorPrivate` `Ctx::SetKey (const PrivateKey &key) noexcept=0;` |
| Parameters (in): | key | the source key object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/decryptor_private_ctx.h" | |
| Description: | Set (deploy) a key to the decryptor private algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

**[SWS_CRYPT_29013]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Compare(ReadOnlyMemRegion expected, std::size_t offset=0) |
| Scope: | class ara::crypto::cryp::DigestService |
| Syntax: | `virtual ara::core::Result<bool> ara::crypto::cryp::Digest` `Service::Compare (ReadOnlyMemRegion expected, std::size_t offset=0)` `const noexcept=0;` |
| Parameters (in): | expected | the memory region containing an expected digest value |
| | offset | position of the first byte in calculated digest for the comparison starting |
| Return value: | ara::core::Result< bool > | true if the expected bytes sequence is identical to first bytes of calculated digest |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |
| | CryptoErrorDomain::kBruteForceRisk | if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context, which was initialized by a key without kAllowSignature permission, but actual size of requested digest is less than 8 bytes (it is a protection from the brute-force attack) |
| Header file: | #include "ara/crypto/cryp/digest_service.h" | |
| Description: | Compare the calculated digest against an expected value. Entire digest value is kept in the context up to next call Start(), therefore any its part can be verified again or extracted. If (full_ digest_size <= offset) \|\| (expected.size() == 0) then return false; else comparison_size = min(expected.size(), (full_digest_size - offset)) bytes. This method can be implemented as "inline" after standartization of function ara::core::memcmp(). |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29012]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetDigestSize() | |
| Scope: | class ara::crypto::cryp::DigestService | |
| Syntax: | `virtual std::size_t ara::crypto::cryp::DigestService::GetDigestSize () const noexcept=0;` | |
| Return value: | std::size_t | size of the full output from this digest-function in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/digest_service.h" | |
| Description: | Get the output digest size. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29015]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsFinished() | |
| Scope: | class ara::crypto::cryp::DigestService | |
| Syntax: | `virtual bool ara::crypto::cryp::DigestService::IsFinished () const noexcept=0;` | |
| Return value: | bool | true if a previously started stream processing was finished by a call of the Finish() or FinishBytes() methods |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/digest_service.h" | |
| Description: | Check current status of the stream processing: finished or no. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29014]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsStarted() | |
| Scope: | class ara::crypto::cryp::DigestService | |
| Syntax: | `virtual bool ara::crypto::cryp::DigestService::IsStarted () const noexcept=0;` | |
| Return value: | bool | true if the processing was start by a call of the Start() methods and was not finished yet |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/digest_service.h" | |
| Description: | Check current status of the stream processing: started or no. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_21002]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetCryptoService() |
| Scope: | class ara::crypto::cryp::EncryptorPublicCtx |
| Syntax: | `virtual CryptoService::Uptr ara::crypto::cryp::EncryptorPublicCtx::Get CryptoService () const noexcept=0;` |
| Return value: | CryptoService::Uptr | – |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/encryptor_public_ctx.h" | |
| Description: | Get CryptoService instance. | |

⌋*(RS_CRYPTO_02006)*

**[SWS_CRYPT_21012]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false) | |
| Scope: | class ara::crypto::cryp::EncryptorPublicCtx | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock (ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept=0;` | |
| Parameters (in): | in | the input data block |
| | suppressPadding | if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/encryptor_public_ctx.h" | |
| Description: | Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutput Size(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppress Padding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! | |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_21013]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false) |
| Scope: | class ara::crypto::cryp::EncryptorPublicCtx |

▽

△

| Syntax: | `template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::Encryptor PublicCtx::ProcessBlock (ReadOnlyMemRegion in, bool suppress Padding=false) const noexcept;` | |
|---|---|---|
| Template param: | Alloc | a custom allocator type of the output container |
| Parameters (in): | in | the input data block |
| | suppressPadding | if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data |
| Return value: | ara::core::Result< ByteVector< Alloc > > | the managed container for output block |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kInsufficient Capacity | if the out.size() is not enough to store the transformation result |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/encryptor_public_ctx.h" | |
| Description: | Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! | |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_21011]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Reset() |
| Scope: | class ara::crypto::cryp::EncryptorPublicCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::EncryptorPublic Ctx::Reset () noexcept=0;` |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/encryptor_public_ctx.h" |
| Description: | Clear the crypto context. |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_21010]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const PublicKey &key) | |
| Scope: | class ara::crypto::cryp::EncryptorPublicCtx | |
| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::EncryptorPublicCtx::SetKey (const PublicKey &key) noexcept=0; | |
| Parameters (in): | key | the source key object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/encryptor_public_ctx.h" | |
| Description: | Set (deploy) a key to the encryptor public algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

**[SWS_CRYPT_29041]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~ExtensionService() |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Syntax: | virtual ara::crypto::cryp::ExtensionService::~ExtensionService () noexcept=default; |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29045]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetActualKeyBitLength() | |
| Scope: | class ara::crypto::cryp::ExtensionService | |
| Syntax: | virtual std::size_t ara::crypto::cryp::ExtensionService::GetActualKey BitLength () const noexcept=0; | |
| Return value: | std::size_t | actual length of a key (now set to the algorithm context) in bits |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/extension_service.h" | |
| Description: | Get actual bit-length of a key loaded to the context. If no key was set to the context yet then 0 is returned. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29047]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetActualKeyCOUID() |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Syntax: | `virtual CryptoObjectUid ara::crypto::cryp::ExtensionService::GetActual KeyCOUID () const noexcept=0;` |
| Return value: | CryptoObjectUid | the COUID of the CryptoObject |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Get the COUID of the key deployed to the context this extension service is attached to. If no key was set to the context yet then an empty COUID (Nil) is returned. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_29046]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetAllowedUsage() |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Syntax: | `virtual AllowedUsageFlags ara::crypto::cryp::ExtensionService::Get AllowedUsage () const noexcept=0;` |
| Return value: | AllowedUsageFlags | a combination of bit-flags that specifies allowed usages of the context |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Get allowed usages of this context (according to the key object attributes loaded to this context). If the context is not initialized by a key object yet then zero (all flags are reset) must be returned. |

⌋*(RS_CRYPTO_02008)*

**[SWS_CRYPT_29044]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetMaxKeyBitLength() |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Syntax: | `virtual std::size_t ara::crypto::cryp::ExtensionService::GetMaxKeyBit Length () const noexcept=0;` |
| Return value: | std::size_t | maximal supported length of the key in bits |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Get maximal supported key length in bits. |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_29043]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetMinKeyBitLength() |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Syntax: | `virtual std::size_t ara::crypto::cryp::ExtensionService::GetMinKeyBit Length () const noexcept=0;` |
| Return value: | std::size_t | minimal supported length of the key in bits |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Get minimal supported key length in bits. |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_29048]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IsKeyBitLengthSupported(std::size_t keyBitLength) |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Syntax: | `virtual bool ara::crypto::cryp::ExtensionService::IsKeyBitLength Supported (std::size_t keyBitLength) const noexcept=0;` |
| Parameters (in): | keyBitLength | length of the key in bits |
| Return value: | bool | true if provided value of the key length is supported by the context |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Verify supportness of specific key length by the context. |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_29049]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IsKeyAvailable() |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Syntax: | `virtual bool ara::crypto::cryp::ExtensionService::IsKeyAvailable () const noexcept=0;` |
| Return value: | bool | FALSE if no key has been set |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Check if a key has been set to this context. |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_30218]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | operator=(const ExtensionService &other) |
| **Scope:** | class ara::crypto::cryp::ExtensionService |
| **Syntax:** | `ExtensionService& ara::crypto::cryp::ExtensionService::operator=` `(const ExtensionService &other)=default;` |
| **Parameters (in):** | other | the other instance |
| **Return value:** | ExtensionService & | *this, containing the contents of other |
| **Header file:** | #include "ara/crypto/cryp/extension_service.h" |
| **Description:** | Copy-assign another ExtensionService to this instance. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30219]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | operator=(ExtensionService &&other) |
| **Scope:** | class ara::crypto::cryp::ExtensionService |
| **Syntax:** | `ExtensionService& ara::crypto::cryp::ExtensionService::operator= (` `ExtensionService &&other)=default;` |
| **Parameters (in):** | other | the other instance |
| **Return value:** | ExtensionService & | *this, containing the contents of other |
| **Header file:** | #include "ara/crypto/cryp/extension_service.h" |
| **Description:** | Move-assign another ExtensionService to this instance. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_21115]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | Finish() | |
| **Scope:** | class ara::crypto::cryp::HashFunctionCtx | |
| **Syntax:** | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >` `ara::crypto::cryp::HashFunctionCtx::Finish () noexcept=0;` | |
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::Byte > > | output data buffer |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if Start() has not been successfully called before. |
| | CryptoErrorDomain::kInvalidUsage Order | if Update() has not been called successfully after the last call to Start() |
| **Header file:** | #include "ara/crypto/cryp/hash_function_ctx.h" | |
| **Description:** | Finish the digest calculation and optionally produce the "signature" object. Only after call of this method the digest can be signed, verified, extracted or compared. | |

⌋*(RS_CRYPTO_02302, RS_CRYPTO_02205)*

### [SWS_CRYPT_21102]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetDigestService() |
| **Scope:** | class ara::crypto::cryp::HashFunctionCtx |
| **Syntax:** | virtual DigestService::Uptr ara::crypto::cryp::HashFunctionCtx::Get DigestService () const noexcept=0; |
| **Return value:** | DigestService::Uptr | – |
| **Exception Safety:** | noexcept |
| **Header file:** | #include "ara/crypto/cryp/hash_function_ctx.h" |
| **Description:** | Get DigestService instance. |

⌋*(RS_CRYPTO_02006)*

### [SWS_CRYPT_21116]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetDigest(std::size_t offset=0) |
| **Scope:** | class ara::crypto::cryp::HashFunctionCtx |
| **Syntax:** | virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::HashFunctionCtx::GetDigest (std::size_t offset=0) const noexcept=0; |
| **Parameters (in):** | offset | position of the first byte of digest that should be placed to the output buffer |
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::Byte > > | number of digest bytes really stored to the output buffer (they are always <= output.size() and denoted below as return_size) |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Errors:** | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |
| **Header file:** | #include "ara/crypto/cryp/hash_function_ctx.h" |
| **Description:** | Get requested part of calculated digest. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.size(), (full_digest_size - offset)) bytes. This method can be implemented as "inline" after standartization of function ara::core::memcpy(). |

⌋*(RS_CRYPTO_02205)*

### [SWS_CRYPT_21117]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetDigest(std::size_t offset=0) |
| **Scope:** | class ara::crypto::cryp::HashFunctionCtx |
| **Syntax:** | template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::HashFunction Ctx::GetDigest (std::size_t offset=0) const noexcept; |
| **Template param:** | Alloc | a custom allocator type of the output container |
| **Parameters (in):** | offset | position of first byte of digest that should be placed to the output buffer |
| **Return value:** | ara::core::Result< ByteVector< Alloc > > | – |

▽

△

| Exception Safety: | noexcept | |
|---|---|---|
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |
| | CryptoErrorDomain::kUsageViolation | if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context initialized by a key without kAllow Signature permission |
| Header file: | #include "ara/crypto/cryp/hash_function_ctx.h" | |
| Description: | Get requested part of calculated digest to pre-reserved managed container. This method sets the size of the output container according to actually saved value. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.capacity(), (full_digest_size - offset)) bytes. | |

⌋*(RS_CRYPTO_02205)*

## [SWS_CRYPT_21118]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Start() | |
| Scope: | class ara::crypto::cryp::HashFunctionCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::HashFunction Ctx::Start () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kMissing Argument | the configured hash function expected an IV |
| Header file: | #include "ara/crypto/cryp/hash_function_ctx.h" | |
| Description: | Initialize the context for a new data stream processing or generation (depending on the primitive) without IV. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_21110]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Start(ReadOnlyMemRegion iv) | |
| Scope: | class ara::crypto::cryp::HashFunctionCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::HashFunction Ctx::Start (ReadOnlyMemRegion iv) noexcept=0;` | |
| Parameters (in): | iv | an optional Initialization Vector (IV) or "nonce" value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false) |

▽

△

| Header file: | #include "ara/crypto/cryp/hash_function_ctx.h" |
|---|---|
| Description: | Initialize the context for a new data stream processing or generation (depending on the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_21111]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Start(const SecretSeed &iv) | |
| Scope: | class ara::crypto::cryp::HashFunctionCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::HashFunctionCtx::Start (const SecretSeed &iv) noexcept=0;` | |
| Parameters (in): | iv | the Initialization Vector (IV) or "nonce" object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation |
| Header file: | #include "ara/crypto/cryp/hash_function_ctx.h" | |
| Description: | Initialize the context for a new data stream processing or generation (depending on the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_21112]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Update(const RestrictedUseObject &in) | |
| Scope: | class ara::crypto::cryp::HashFunctionCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::HashFunctionCtx::Update (const RestrictedUseObject &in) noexcept=0;` | |
| Parameters (in): | in | a part of input message that should be processed |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNotStarted | if the digest calculation was not initiated by a call of the Start() method |
| Header file: | #include "ara/crypto/cryp/hash_function_ctx.h" | |
| Description: | Update the digest calculation context by a new part of the message. This method is dedicated for cases then the RestrictedUseObject is a part of the "message". | |

⌋*(RS_CRYPTO_02302)*

### [SWS_CRYPT_21113]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | Update(ReadOnlyMemRegion in) |
| **Scope:** | class ara::crypto::cryp::HashFunctionCtx |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::HashFunction`<br>`Ctx::Update (ReadOnlyMemRegion in) noexcept=0;` |
| **Parameters (in):** | in | a part of the input message that should be processed |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| **Header file:** | #include "ara/crypto/cryp/hash_function_ctx.h" | |
| **Description:** | Update the digest calculation context by a new part of the message. | |

⌋*(RS_CRYPTO_02302)*

### [SWS_CRYPT_21114]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | Update(std::uint8_t in) |
| **Scope:** | class ara::crypto::cryp::HashFunctionCtx |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::HashFunction`<br>`Ctx::Update (std::uint8_t in) noexcept=0;` |
| **Parameters (in):** | in | a byte value that is a part of input message |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| **Header file:** | #include "ara/crypto/cryp/hash_function_ctx.h" | |
| **Description:** | Update the digest calculation context by a new part of the message. This method is convenient for processing of constant tags. | |

⌋*(RS_CRYPTO_02302)*

### [SWS_CRYPT_21312]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | AgreeKey(const PublicKey &otherSideKey, CryptoAlgId targetAlgId, AllowedUsageFlags allowedUsage, ara::core::Optional< ReadOnlyMemRegion > salt, ara::core::Optional< ReadOnlyMemRegion > ctxLabel) |
| **Scope:** | class ara::crypto::cryp::KeyAgreementPrivateCtx |
| **Syntax:** | `virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::Key`<br>`AgreementPrivateCtx::AgreeKey (const PublicKey &otherSideKey, Crypto`<br>`AlgId targetAlgId, AllowedUsageFlags allowedUsage,`<br>`ara::core::Optional< ReadOnlyMemRegion > salt, ara::core::Optional<`<br>`ReadOnlyMemRegion > ctxLabel) const noexcept=0;` |
| **Parameters (in):** | otherSideKey | the public key of the other side of the Key-Agreement |

▽

$\triangle$

| | targetAlgId | identifier of the target symmetric algorithm (also defines a target key-length) |
|---|---|---|
| | allowedUsage | the allowed usage scope of the target key |
| | salt | an optional salt value (if used, it should be unique for each instance of the target key) |
| | ctxLabel | an optional application specific "context label" (it can identify purpose of the target key and/or communication parties) |
| **Return value:** | ara::core::Result< Symmetric Key::Uptrc > | a unique pointer to SymmetricKey object, which contains the computed shared secret or key material produced by the Key-Agreement algorithm |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrc::kUninitializedContext | if the context was not initialized by a key value |
| | CryptoErrc::kIncompatibleObject | if the public and private keys correspond to different algorithms |
| **Header file:** | #include "ara/crypto/cryp/key_agreement_private_ctx.h" | |
| **Description:** | Produce a common symmetric key via execution of the key-agreement algorithm between this private key and a public key of another side. Produced SymmetricKey object has following attributes: session, non-exportable. This method can be used for direct production of the target key, without creation of the intermediate SecretSeed object. | |

⌋*(RS_CRYPTO_02115)*

### [SWS_CRYPT_21311]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | AgreeSeed(const PublicKey &otherSideKey, ara::core::Optional< AllowedUsageFlags > allowed Usage) | |
| **Scope:** | class ara::crypto::cryp::KeyAgreementPrivateCtx | |
| **Syntax:** | `virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::Key AgreementPrivateCtx::AgreeSeed (const PublicKey &otherSideKey, ara::core::Optional< AllowedUsageFlags > allowedUsage) const noexcept=0;` | |
| **Parameters (in):** | otherSideKey | the public key of the other side of the Key-Agreement |
| | allowedUsage | the allowed usage scope of the target seed |
| **Return value:** | ara::core::Result< SecretSeed::Uptrc > | unique pointer to SecretSeed object, which contains the key material produced by the Key-Agreement algorithm |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrc::kUninitializedContext | if the context was not initialized by a key value |
| | CryptoErrc::kIncompatibleObject | if the public and private keys correspond to different algorithms |
| **Header file:** | #include "ara/crypto/cryp/key_agreement_private_ctx.h" | |
| **Description:** | Produce a common secret seed via execution of the key-agreement algorithm between this private key and a public key of another side. Produced SecretSeed object has following attributes: session, non-exportable, AlgID (this Key-Agreement Algorithm ID). | |

⌋*(RS_CRYPTO_02007)*

## [SWS_CRYPT_21302]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetExtensionService() |
| Scope: | class ara::crypto::cryp::KeyAgreementPrivateCtx |
| Syntax: | virtual ExtensionService::Uptr ara::crypto::cryp::KeyAgreementPrivateCtx::GetExtensionService () const noexcept=0; |
| Return value: | ExtensionService::Uptr | – |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/key_agreement_private_ctx.h" |
| Description: | Get ExtensionService instance. |

⌋(*RS_CRYPTO_02006*)

## [SWS_CRYPT_21314]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Reset() |
| Scope: | class ara::crypto::cryp::KeyAgreementPrivateCtx |
| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::KeyAgreementPrivateCtx::Reset () noexcept=0; |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/key_agreement_private_ctx.h" |
| Description: | Clear the crypto context. |

⌋(*RS_CRYPTO_02108*)

## [SWS_CRYPT_21313]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const PrivateKey &key) | |
| Scope: | class ara::crypto::cryp::KeyAgreementPrivateCtx | |
| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::KeyAgreementPrivateCtx::SetKey (const PrivateKey &key) noexcept=0; | |
| Parameters (in): | key | the source key object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kIncompatibleObject | if the provided key object is incompatible with this private key context |
| | CryptoErrc::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/key_agreement_private_ctx.h" | |
| Description: | Set (deploy) a key to the key agreement private algorithm context. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02003*)

### [SWS_CRYPT_21315]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKDF(const KeyDerivationFunctionCtx &kdf) | |
| Scope: | class ara::crypto::cryp::KeyAgreementPrivateCtx | |
| Syntax: | virtual void ara::crypto::cryp::KeyAgreementPrivateCtx::SetKDF (const KeyDerivationFunctionCtx &kdf) const noexcept=0; | |
| Parameters (in): | kdf | the KeyDerivationFunctionCtx that shall be used to derive the final SymmetricKey or SecretSeed. |
| Return value: | None | |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_agreement_private_ctx.h" | |
| Description: | This interface may be used to provide a KDF in case no KDF was specified during context creation or the default parameters of the specified KDF are insufficient or the specified KDF must be replaced. | |

⌋(RS_CRYPTO_02115)

### [SWS_CRYPT_21412]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | DecapsulateKey(ReadOnlyMemRegion input, CryptoAlgId keyingDataAlgId, KeyDerivation FunctionCtx &kdf, CryptoAlgId kekAlgId, ara::core::Optional< AllowedUsageFlags > allowed Usage) | |
| Scope: | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx | |
| Syntax: | virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::Key DecapsulatorPrivateCtx::DecapsulateKey (ReadOnlyMemRegion input, CryptoAlgId keyingDataAlgId, KeyDerivationFunctionCtx &kdf, CryptoAlg Id kekAlgId, ara::core::Optional< AllowedUsageFlags > allowedUsage) const noexcept=0; | |
| Parameters (in): | input | an input buffer (its size should be equal Get EncapsulatedSize() bytes) |
| | keyingDataAlgId | algorithm ID of the returned symmetric key |
| | kdf | a context of a key derivation function, which should be used for KEK production |
| | kekAlgId | an algorithm ID of the KEK |
| | allowedUsage | the allowed usage scope of the returned symmetric key object (default = kAllowKdfMaterialAnyUsage) |
| Return value: | ara::core::Result< Symmetric Key::Uptrc > | unique smart pointer of the symmetric key object instantiated from the decapsulated keying data |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kUninitializedContext | if the context was not initialized by a private key value |
| | CryptoErrc::kInvalidArgument | if kekAlgId or kdf are incompatible with this context |
| | CryptoErrc::kInvalidInputSize | if this context does not support the size of input |
| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" | |
| Description: | Decapsulate the keying data to be used for subsequent processing (e.g. secure communication). Produced SymmetricKey object has following attributes: session, non-exportable. | |

⌋(RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115)

**[SWS_CRYPT_21411]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | DecapsulateSeed(ReadOnlyMemRegion input, ara::core::Optional< AllowedUsageFlags > allowedUsage) | |
| Scope: | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx | |
| Syntax: | `virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::Key DecapsulatorPrivateCtx::DecapsulateSeed (ReadOnlyMemRegion input, ara::core::Optional< AllowedUsageFlags > allowedUsage) const noexcept=0;` | |
| Parameters (in): | input | a buffer with the encapsulated seed (its size should be equal GetEncapsulatedSize() bytes) |
| | allowedUsage | the allowed usage scope of the target seed (default = kAllowKdfMaterialAnyUsage) |
| Return value: | ara::core::Result< SecretSeed::Uptrc > | unique smart pointer to SecretSeed object, which keeps the key material decapsulated from the input buffer |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kUninitializedContext | if the context was not initialized by a private key value |
| | CryptoErrc::kInvalidInputSize | if this context does not support the size of input |
| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" | |
| Description: | Decapsulate key material. Produced SecretSeed object has following attributes: session, non-exportable, AlgID = this KEM AlgID. | |

⌋*(RS_CRYPTO_02007)*

**[SWS_CRYPT_21416]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetEncapsulatedSize() | |
| Scope: | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx | |
| Syntax: | `virtual std::size_t ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Get EncapsulatedSize () const noexcept=0;` | |
| Return value: | std::size_t | size of the encapsulated data block in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" | |
| Description: | Get fixed size of the encapsulated data block. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_21402]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetExtensionService() | |
| Scope: | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx | |
| Syntax: | `virtual ExtensionService::Uptr ara::crypto::cryp::KeyDecapsulator PrivateCtx::GetExtensionService () const noexcept=0;` | |
| Return value: | ExtensionService::Uptr | – |
| Exception Safety: | noexcept | |

▽

△

| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" |
|---|---|
| Description: | Get ExtensionService instance. |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_21415]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetKekEntropy() | |
| Scope: | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx | |
| Syntax: | `virtual std::size_t ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Get KekEntropy () const noexcept=0;` | |
| Return value: | std::size_t | entropy of the KEK material in bits |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" | |
| Description: | Get entropy (bit-length) of the key encryption key (KEK) material. For RSA system the returned value corresponds to the length of module N (minus 1). For DH-like system the returned value corresponds to the length of module q (minus 1). | |

⌋*(RS_CRYPTO_02309)*

## [SWS_CRYPT_21414]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Reset() | |
| Scope: | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::KeyDecapsulator PrivateCtx::Reset () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" | |
| Description: | Clear the crypto context. | |

⌋*(RS_CRYPTO_02108)*

## [SWS_CRYPT_21413]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const PrivateKey &key) | |
| Scope: | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::KeyDecapsulator PrivateCtx::SetKey (const PrivateKey &key) noexcept=0;` | |
| Parameters (in): | key | the source key object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |

▽

△

| Errors: | CryptoErrc::kIncompatibleObject | if the provided key object is incompatible with this private key context |
| | CryptoErrc::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" | |
| Description: | Set (deploy) a key to the key decapsulator private algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

## [SWS_CRYPT_21512]{DRAFT} ⌈

| Kind: | function | |
| Symbol: | AddSalt(ReadOnlyMemRegion salt) | |
| Scope: | class ara::crypto::cryp::KeyDerivationFunctionCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::KeyDerivation FunctionCtx::AddSalt (ReadOnlyMemRegion salt) noexcept=0;` | |
| Parameters (in): | salt | a salt value (if used, it should be unique for each instance of the target key) |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| Description: | Add a salt value stored in a (non-secret) ReadOnlyMemRegion. | |

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_-02111)*

## [SWS_CRYPT_21513]{DRAFT} ⌈

| Kind: | function | |
| Symbol: | AddSecretSalt(const SecretSeed &salt) | |
| Scope: | class ara::crypto::cryp::KeyDerivationFunctionCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::KeyDerivation FunctionCtx::AddSecretSalt (const SecretSeed &salt) noexcept=0;` | |
| Parameters (in): | salt | a salt value (if used, it should be unique for each instance of the target key) |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| Description: | Add a secret salt value stored in a SecretSeed object. | |

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_-02111)*

**[SWS_CRYPT_21514]**{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | ConfigIterations(std::uint32_t iterations=0) |
| *Scope:* | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| *Syntax:* | `virtual std::uint32_t ara::crypto::cryp::KeyDerivationFunction Ctx::ConfigIterations (std::uint32_t iterations=0) noexcept=0;` |
| *Parameters (in):* | iterations | the required number of iterations of the base function (0 means implementation default number) |
| *Return value:* | std::uint32_t | actual number of the iterations configured in the context now (after this method call) |
| *Exception Safety:* | noexcept |
| *Thread Safety:* | Thread-safe |
| *Header file:* | #include "ara/crypto/cryp/key_derivation_function_ctx.h" |
| *Description:* | Configure the number of iterations that will be applied by default. Implementation can restrict minimal and/or maximal value of the iterations number. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_21515]**{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | DeriveKey(bool isSession=true, bool isExportable=false) |
| *Scope:* | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| *Syntax:* | `virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::Key DerivationFunctionCtx::DeriveKey (bool isSession=true, bool is Exportable=false) const noexcept=0;` |
| *Parameters (in):* | isSession | the "session" (or "temporary") attribute for the target key (if true) |
| | isExportable | the exportability attribute for the target key (if true) |
| *Return value:* | ara::core::Result< Symmetric Key::Uptrc > | unique smart pointer to the created instance of derived symmetric key |
| *Exception Safety:* | noexcept |
| *Thread Safety:* | Thread-safe |
| *Errors:* | CryptoErrorDomain::kUninitialized Context | if the context was not sufficiently initialized |
| *Header file:* | #include "ara/crypto/cryp/key_derivation_function_ctx.h" |
| *Description:* | Derive a symmetric key from the provided key material and provided context configuration. |

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_-02111, RS_CRYPTO_02115)*

**[SWS_CRYPT_21516]**{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | DeriveSeed(bool isSession=true, bool isExportable=false) |
| *Scope:* | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| *Syntax:* | `virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::Key DerivationFunctionCtx::DeriveSeed (bool isSession=true, bool is Exportable=false) const noexcept=0;` |
| *Parameters (in):* | isSession | the "session" (or "temporary") attribute for the target key (if true) |

▽

△

| | isExportable | the exportability attribute for the target key (if true) |
|---|---|---|
| **Return value:** | ara::core::Result< SecretSeed::Uptrc > | unique smart pointer to the created SecretSeed object |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUninitialized Context | if the context was not sufficiently initialized |
| **Header file:** | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| **Description:** | Derive a "slave" key material (secret seed) from the provided "master" key material and provided context configuration. | |

⌋*(RS_CRYPTO_02007)*

## [SWS_CRYPT_21524]{DRAFT} ⌈

| **Kind:** | function |
|---|---|
| **Symbol:** | Reset() |
| **Scope:** | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::KeyDerivation FunctionCtx::Reset () noexcept=0;` |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/cryp/key_derivation_function_ctx.h" |
| **Description:** | Clear the crypto context. |

⌋*(RS_CRYPTO_02108)*

## [SWS_CRYPT_21517]{DRAFT} ⌈

| **Kind:** | function |
|---|---|
| **Symbol:** | GetExtensionService() |
| **Scope:** | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| **Syntax:** | `virtual ExtensionService::Uptr ara::crypto::cryp::KeyDerivation FunctionCtx::GetExtensionService () const noexcept=0;` |
| **Return value:** | ExtensionService::Uptr | – |
| **Exception Safety:** | noexcept |
| **Header file:** | #include "ara/crypto/cryp/key_derivation_function_ctx.h" |
| **Description:** | Get ExtensionService instance. |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_21519]{DRAFT} ⌈

| **Kind:** | function |
|---|---|
| **Symbol:** | GetKeyIdSize() |
| **Scope:** | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| **Syntax:** | `virtual std::size_t ara::crypto::cryp::KeyDerivationFunctionCtx::Get KeyIdSize () const noexcept=0;` |

▽

△

| | | |
|---|---|---|
| *Return value:* | std::size_t | size of the key ID in bytes the @c Init() call. |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| *Description:* | Get the fixed size of the target key ID required by diversification algorithm. Returned value is constant for each instance of the interface, i.e. independent from configuration by.<br><br>Returned value is constant for this instance of the key derivation context, i.e. independent from configuration by | |

⌋*(RS_CRYPTO_02103)*

## [SWS_CRYPT_21520]{DRAFT} ⌈

| | | |
|---|---|---|
| *Kind:* | function | |
| *Symbol:* | GetTargetAlgId() | |
| *Scope:* | class ara::crypto::cryp::KeyDerivationFunctionCtx | |
| *Syntax:* | `virtual AlgId ara::crypto::cryp::KeyDerivationFunctionCtx::GetTarget AlgId () const noexcept=0;` | |
| *Return value:* | AlgId | the symmetric algorithm ID of the target key, configured by the last call of the Init() method returned. |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| *Description:* | Get the symmetric algorithm ID of target (slave) key. If the context was not configured yet by a call of the Init() method then kAlgIdUndefined should be. | |

⌋*(RS_CRYPTO_02103)*

## [SWS_CRYPT_21521]{DRAFT} ⌈

| | | |
|---|---|---|
| *Kind:* | function | |
| *Symbol:* | GetTargetAllowedUsage() | |
| *Scope:* | class ara::crypto::cryp::KeyDerivationFunctionCtx | |
| *Syntax:* | `virtual AllowedUsageFlags ara::crypto::cryp::KeyDerivationFunction Ctx::GetTargetAllowedUsage () const noexcept=0;` | |
| *Return value:* | AllowedUsageFlags | allowed key usage bit-flags of target keys |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| *Description:* | Get allowed key usage of target (slave) key. The returned value depends on the source key-material allowed usage flags and the argument allowedUsage of last call of the Init() method. If the context has not yet been configured by a call of the Init() method, the allowed usage flags of the source key-material shall be returned. If the context has not yet been configured by a call of the Init() method and no source key-material has been set either, kAllow KdfMaterialAnyUsage shall be returned. | |

⌋*(RS_CRYPTO_02008)*

### [SWS_CRYPT_21522]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetTargetKeyBitLength() |
| **Scope:** | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| **Syntax:** | `virtual std::size_t ara::crypto::cryp::KeyDerivationFunctionCtx::Get` `TargetKeyBitLength () const noexcept=0;` |
| **Return value:** | std::size_t | the length of target (diversified) key in bits the @c Init() calls. |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| **Description:** | Get the bit-length of target (diversified) keys. Returned value is configured by the context factory method, i.e. independent from configuration by. | |

⌋*(RS_CRYPTO_02103)*

### [SWS_CRYPT_21523]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | Init(ReadOnlyMemRegion targetKeyId, AlgId targetAlgId=kAlgIdAny, AllowedUsageFlags allowedUsage=kAllowKdfMaterialAnyUsage, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion()) |
| **Scope:** | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::KeyDerivation` `FunctionCtx::Init (ReadOnlyMemRegion targetKeyId, AlgId targetAlgId=k` `AlgIdAny, AllowedUsageFlags allowedUsage=kAllowKdfMaterialAnyUsage,` `ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion()) noexcept=0;` |
| **Parameters (in):** | targetKeyId | ID of the target key |
| | targetAlgId | the identifier of the target symmetric crypto algorithm |
| | allowedUsage | bit-flags that define a list of allowed transformations' types in which the target key may be used |
| | ctxLabel | an optional application specific "context label" (this can identify the purpose of the target key and/or communication parties) |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kIncompatible Arguments | if targetAlgId specifies a cryptographic algorithm different from a symmetric one with key length equal to GetTargetKeyBitLength(); |
| | CryptoErrorDomain::kUsageViolation | if allowedUsage specifies more usages of the derived key-material than the source key-material, i.e. usage of the derived key-material may not be expanded beyond what the source key-material allows |
| **Header file:** | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| **Description:** | Initialize this context by setting at least the target key ID. The byte sequence provided via argument ctxLabel can include a few fields with different meaning separated by single 0x00 byte. If (targetAlgId == kAlgIdAny) then a diversified key can be loaded to any symmetric context that supports the same key length (if the "allowed usage" flags are also satisfied)! | |

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_-02111, RS_CRYPTO_02115)*

## [SWS_CRYPT_21525]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetSourceKeyMaterial(const RestrictedUseObject &sourceKM) | |
| Scope: | class ara::crypto::cryp::KeyDerivationFunctionCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::KeyDerivation FunctionCtx::SetSourceKeyMaterial (const RestrictedUseObject &source KM) noexcept=0;` | |
| Parameters (in): | sourceKM | the source key-material |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if deriving a key is prohibited by the "allowed usage" restrictions of the provided source key-material |
| | CryptoErrorDomain::kBruteForceRisk | if key length of the sourceKm is below of an internally defined limitation |
| Header file: | #include "ara/crypto/cryp/key_derivation_function_ctx.h" | |
| Description: | Set (deploy) key-material to the key derivation algorithm context. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02003*)

## [SWS_CRYPT_21818]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetEncapsulatedSize() | |
| Scope: | class ara::crypto::cryp::KeyEncapsulatorPublicCtx | |
| Syntax: | `virtual std::size_t ara::crypto::cryp::KeyEncapsulatorPublicCtx::Get EncapsulatedSize () const noexcept=0;` | |
| Return value: | std::size_t | size of the encapsulated data block in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" | |
| Description: | Get fixed size of the encapsulated data block. | |

⌋(*RS_CRYPTO_02309*)

## [SWS_CRYPT_21802]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetExtensionService() | |
| Scope: | class ara::crypto::cryp::KeyEncapsulatorPublicCtx | |
| Syntax: | `virtual ExtensionService::Uptr ara::crypto::cryp::KeyEncapsulator PublicCtx::GetExtensionService () const noexcept=0;` | |
| Return value: | ExtensionService::Uptr | – |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" | |
| Description: | Get ExtensionService instance. | |

⌋(*RS_CRYPTO_02006*)

### [SWS_CRYPT_21817]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetKekEntropy() | |
| Scope: | class ara::crypto::cryp::KeyEncapsulatorPublicCtx | |
| Syntax: | `virtual std::size_t ara::crypto::cryp::KeyEncapsulatorPublicCtx::Get`<br>`KekEntropy () const noexcept=0;` | |
| Return value: | std::size_t | entropy of the KEK material in bits |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" | |
| Description: | Get entropy (bit-length) of the key encryption key (KEK) material. For RSA system the returned value corresponds to the length of module N (minus 1). For DH-like system the returned value corresponds to the length of module q (minus 1). | |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_21810]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | AddKeyingData(const RestrictedUseObject &keyingData) | |
| Scope: | class ara::crypto::cryp::KeyEncapsulatorPublicCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::KeyEncapsulator`<br>`PublicCtx::AddKeyingData (const RestrictedUseObject &keyingData)`<br>`noexcept=0;` | |
| Parameters (in): | keyingData | the payload to be protected |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kUsageViolation | if the keyingData cannot be exported due to Crypto Object::IsExportable() returning FALSE |
| | CryptoErrc::kIncompatibleObject | if the keyingData belongs to a different Crypto Provider |
| | CryptoErrc::kInvalidInputSize | if this context does not support the size of the keying Data |
| Header file: | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" | |
| Description: | Add the content to be encapsulated (payload) according to RFC 5990 ("keying data"). At the moment only SymmetricKey and SecretSeed objects are supported. | |

⌋*(RS_CRYPTO_02007)*

### [SWS_CRYPT_21813]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Encapsulate(KeyDerivationFunctionCtx &kdf, CryptoAlgId kekAlgId) | |
| Scope: | class ara::crypto::cryp::KeyEncapsulatorPublicCtx | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::cryp::KeyEncapsulatorPublicCtx::Encapsulate (Key`<br>`DerivationFunctionCtx &kdf, CryptoAlgId kekAlgId) const noexcept=0;` | |
| Parameters (in): | kdf | a context of a key derivation function, which should be used for the target KEK production |
| | kekAlgId | an algorithm ID of the target KEK |

▽

△

| | | |
|---|---|---|
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::Byte > > | the encapsulated data as a byte-vector |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrc::kUninitializedContext | if the context was not initialized by a public key value |
| | CryptoErrc::kInvalidArgument | if kekAlgId or kdf are incompatible with this context |
| **Header file:** | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" | |
| **Description:** | Encapsulate the last set keying-data. | |

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115)*

## [SWS_CRYPT_21816]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | Reset() |
| **Scope:** | class ara::crypto::cryp::KeyEncapsulatorPublicCtx |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::KeyEncapsulator`<br>`PublicCtx::Reset () noexcept=0;` |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" |
| **Description:** | Clear the crypto context. |

⌋*(RS_CRYPTO_02108)*

## [SWS_CRYPT_21815]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | SetKey(const PublicKey &key) | |
| **Scope:** | class ara::crypto::cryp::KeyEncapsulatorPublicCtx | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::KeyEncapsulator`<br>`PublicCtx::SetKey (const PublicKey &key) noexcept=0;` | |
| **Parameters (in):** | key | the source key object |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrc::kIncompatibleObject | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrc::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| **Header file:** | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" | |
| **Description:** | Set (deploy) a key to the key encapsulator public algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

## [SWS_CRYPT_22119]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Check(const Signature &expected) | |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| Syntax: | `virtual ara::core::Result<bool> ara::crypto::cryp::MessageAuthnCode Ctx::Check (const Signature &expected) const noexcept=0;` | |
| Parameters (in): | expected | the signature object containing an expected digest value |
| Return value: | ara::core::Result< bool > | true if value and meta-information of the provided "signature" object is identical to calculated digest and current configuration of the context respectively; but false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |
| | CryptoErrorDomain::kIncompatible Object | if the provided "signature" object was produced by another crypto primitive type |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| Description: | Check the calculated digest against an expected "signature" object. Entire digest value is kept in the context up to next call Start(), therefore it can be verified again or extracted. This method can be implemented as "inline" after standartization of function ara::core::memcmp(). | |

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02204)*

## [SWS_CRYPT_22115]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Finish(bool makeSignatureObject=false) | |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| Syntax: | `virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::Message AuthnCodeCtx::Finish (bool makeSignatureObject=false) noexcept=0;` | |
| Parameters (in): | makeSignatureObject | if this argument is true then the method will also produce the signature object |
| Return value: | ara::core::Result< Signature::Uptrc > | unique smart pointer to created signature object, if (makeSignatureObject == true) or nullptr if (make SignatureObject == false) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| | CryptoErrorDomain::kUsageViolation | if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context initialized by a key without kAllow Signature permission, but (makeSignatureObject == true) |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| Description: | Finish the digest calculation and optionally produce the "signature" object. Only after call of this method the digest can be signed, verified, extracted or compared! If the signature object produced by a keyed MAC/HMAC/AE/AEAD algorithm then the dependence COUID of the "signature" should be set to COUID of used symmetric key. | |

⌋*(RS_CRYPTO_02302, RS_CRYPTO_02203)*

### [SWS_CRYPT_22102]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetDigestService() |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx |
| Syntax: | `virtual DigestService::Uptr ara::crypto::cryp::MessageAuthnCode`<br>`Ctx::GetDigestService () const noexcept=0;` |
| Return value: | DigestService::Uptr | – |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" |
| Description: | Get DigestService instance. |

⌋*(RS_CRYPTO_02006)*

### [SWS_CRYPT_22116]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetDigest(std::size_t offset=0) | |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::cryp::MessageAuthnCodeCtx::GetDigest (std::size_t`<br>`offset=0) const noexcept=0;` | |
| Parameters (in): | offset | position of the first byte of digest that should be placed to the output buffer |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | number of digest bytes really stored to the output buffer (they are always <= output.size() and denoted below as return_size) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |
| | CryptoErrorDomain::kUsageViolation | if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context initialized by a key without kAllow Signature permission |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| Description: | Get requested part of calculated digest to existing memory buffer. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.size(), (full_digest_size - offset)) bytes. This method can be implemented as "inline" after standartization of function ara::core::memcpy(). | |

⌋*(RS_CRYPTO_02203)*

### [SWS_CRYPT_22117]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetDigest(std::size_t offset=0) | |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| Syntax: | `template <typename Alloc = <implementation-defined>>`<br>`ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::MessageAuthn`<br>`CodeCtx::GetDigest (std::size_t offset=0) const noexcept;` | |
| Template param: | Alloc | a custom allocator type of the output container |

▽

△

| Parameters (in): | offset | position of first byte of digest that should be placed to the output buffer |
|---|---|---|
| Return value: | ara::core::Result< ByteVector< Alloc > > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kProcessingNot Finished | if the digest calculation was not finished by a call of the Finish() method |
| | CryptoErrorDomain::kUsageViolation | if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context initialized by a key without kAllow Signature permission |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| Description: | Get requested part of calculated digest to pre-reserved managed container. This method sets the size of the output container according to actually saved value. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.capacity(), (full_digest_size - offset)) bytes. | |

⌋*(RS_CRYPTO_02203)*

## [SWS_CRYPT_22120]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Reset() |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::MessageAuthnCode Ctx::Reset () noexcept=0;` |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| Description: | Clear the crypto context. | |

⌋*(RS_CRYPTO_02108)*

## [SWS_CRYPT_22118]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kMac Generate) |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::MessageAuthnCode Ctx::SetKey (const SymmetricKey &key, CryptoTransform transform=Crypto Transform::kMacGenerate) noexcept=0;` |
| Parameters (in): | key | the source key object |
| | transform | the "direction" indicator: deploy the key for direct transformation (if true) or for reverse one (if false) |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |

▽

△

| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
|---|---|---|
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| Description: | Set (deploy) a key to the message authn code algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

## [SWS_CRYPT_22110]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Start(ReadOnlyMemRegion iv=ReadOnlyMemRegion()) | |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::MessageAuthnCode Ctx::Start (ReadOnlyMemRegion iv=ReadOnlyMemRegion()) noexcept=0;` | |
| Parameters (in): | iv | an optional Initialization Vector (IV) or "nonce" value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by deploying a key |
| | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false) |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| Description: | Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_22111]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Start(const SecretSeed &iv) | |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::MessageAuthnCode Ctx::Start (const SecretSeed &iv) noexcept=0;` | |
| Parameters (in): | iv | the Initialization Vector (IV) or "nonce" object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by deploying a key |

▽

△

| | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
|---|---|---|
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation |
| | CryptoErrorDomain::kUsageViolation | if this transformation type is prohibited by the "allowed usage" restrictions of the provided Secret Seed object |
| **Header file:** | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| **Description:** | Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_22112]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | Update(const RestrictedUseObject &in) | |
| **Scope:** | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::MessageAuthnCode` `Ctx::Update (const RestrictedUseObject &in) noexcept=0;` | |
| **Parameters (in):** | in | a part of input message that should be processed |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| **Header file:** | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| **Description:** | Update the digest calculation context by a new part of the message. This method is dedicated for cases then the RestrictedUseObject is a part of the "message". | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_22113]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | Update(ReadOnlyMemRegion in) | |
| **Scope:** | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::MessageAuthnCode` `Ctx::Update (ReadOnlyMemRegion in) noexcept=0;` | |
| **Parameters (in):** | in | a part of the input message that should be processed |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| **Header file:** | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| **Description:** | Update the digest calculation context by a new part of the message. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_22114]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | Update(std::uint8_t in) | |
| **Scope:** | class ara::crypto::cryp::MessageAuthnCodeCtx | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::cryp::MessageAuthnCode Ctx::Update (std::uint8_t in) noexcept=0;` | |
| **Parameters (in):** | in | a byte value that is a part of input message |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kProcessingNot Started | if the digest calculation was not initiated by a call of the Start() method |
| **Header file:** | #include "ara/crypto/cryp/message_authn_code_ctx.h" | |
| **Description:** | Update the digest calculation context by a new part of the message. This method is convenient for processing of constant tags. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_22210]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | GetExtensionService() | |
| **Scope:** | class ara::crypto::cryp::MsgRecoveryPublicCtx | |
| **Syntax:** | `virtual ExtensionService::Uptr ara::crypto::cryp::MsgRecoveryPublic Ctx::GetExtensionService () const noexcept=0;` | |
| **Return value:** | ExtensionService::Uptr | – |
| **Exception Safety:** | noexcept | |
| **Header file:** | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" | |
| **Description:** | Get ExtensionService instance. | |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_22213]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | GetMaxInputSize(bool suppressPadding=false) | |
| **Scope:** | class ara::crypto::cryp::MsgRecoveryPublicCtx | |
| **Syntax:** | `virtual std::size_t ara::crypto::cryp::MsgRecoveryPublicCtx::GetMax InputSize (bool suppressPadding=false) const noexcept=0;` | |
| **Parameters (in):** | suppressPadding | if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only |
| **Return value:** | std::size_t | maximum size of the input data block in bytes |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" | |
| **Description:** | Get maximum expected size of the input data block. If (IsEncryption() == false) then a value returned by this method is independent from the suppressPadding argument and it will be equal to the block size. | |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_22214]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | GetMaxOutputSize(bool suppressPadding=false) |
| *Scope:* | class ara::crypto::cryp::MsgRecoveryPublicCtx |
| *Syntax:* | `virtual std::size_t ara::crypto::cryp::MsgRecoveryPublicCtx::GetMax`<br>`OutputSize (bool suppressPadding=false) const noexcept=0;` |

| *Parameters (in):* | suppressPadding | if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only |
|---|---|---|
| *Return value:* | std::size_t | maximum size of the output data block in bytes |

| | |
|---|---|
| *Exception Safety:* | noexcept |
| *Thread Safety:* | Thread-safe |
| *Header file:* | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" |
| *Description:* | Get maximum possible size of the output data block. If (IsEncryption() == true) then a value returned by this method is independent from the suppressPadding argument and will be equal to the block size. |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_22215]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | DecodeAndVerify(ReadOnlyMemRegion in) |
| *Scope:* | class ara::crypto::cryp::MsgRecoveryPublicCtx |
| *Syntax:* | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::cryp::MsgRecoveryPublicCtx::DecodeAndVerify (ReadOnlyMem`<br>`Region in) const noexcept=0;` |

| *Parameters (in):* | in | the input data block |
|---|---|---|
| *Return value:* | ara::core::Result< ara::core::Vector< ara::core::Byte > > | the output buffer actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content |

| | |
|---|---|
| *Exception Safety:* | noexcept |
| *Thread Safety:* | Thread-safe |

| *Errors:* | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
|---|---|---|
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |

| | |
|---|---|
| *Header file:* | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" |
| *Description:* | Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutput Size(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppress Padding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! |

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_22216]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | DecodeAndVerify(ReadOnlyMemRegion in) | |
| Scope: | class ara::crypto::cryp::MsgRecoveryPublicCtx | |
| Syntax: | `template <typename Alloc = <implementation-defined>>`<br>`ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::MsgRecovery`<br>`PublicCtx::DecodeAndVerify (ReadOnlyMemRegion in) const noexcept;` | |
| Template param: | Alloc | a custom allocator type of the output container |
| Parameters (in): | in | the input data block |
| Return value: | ara::core::Result< ByteVector< Alloc > > | the managed container for output block |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" | |
| Description: | Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! | |

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_22212]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Reset() | |
| Scope: | class ara::crypto::cryp::MsgRecoveryPublicCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::MsgRecoveryPublic`<br>`Ctx::Reset () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" | |
| Description: | Clear the crypto context. | |

⌋*(RS_CRYPTO_02108)*

**[SWS_CRYPT_22211]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetKey(const PublicKey &key) |
| Scope: | class ara::crypto::cryp::MsgRecoveryPublicCtx |

▽

△

| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::MsgRecoveryPublic Ctx::SetKey (const PublicKey &key) noexcept=0; | |
|---|---|---|
| Parameters (in): | key | the source key object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" | |
| Description: | Set (deploy) a key to the msg recovery public algorithm context. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02003*)

## [SWS_CRYPT_22511]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetPublicKey() | |
| Scope: | class ara::crypto::cryp::PrivateKey | |
| Syntax: | virtual ara::core::Result<PublicKey::Uptrc> ara::crypto::cryp::Private Key::GetPublicKey () const noexcept=0; | |
| Return value: | ara::core::Result< PublicKey::Uptrc > | unique smart pointer to the public key correspondent to this private key |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/private_key.h" | |
| Description: | Get the public key correspondent to this private key. | |

⌋(*RS_CRYPTO_02108*, *RS_CRYPTO_02115*)

## [SWS_CRYPT_22711]{OBSOLETE} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CheckKey(bool strongCheck=true) | |
| Scope: | class ara::crypto::cryp::PublicKey | |
| Syntax: | virtual bool ara::crypto::cryp::PublicKey::CheckKey (bool strong Check=true) const noexcept=0; | |
| Parameters (in): | strongCheck | the severeness flag that indicates type of the required check: strong (if true) or fast (if false) |
| Return value: | bool | true if the key is correct |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/public_key.h" | |
| Description: | Check the key for its correctness. | |

⌋(*RS_CRYPTO_02202*)

## [SWS_CRYPT_22712]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | HashPublicKey(HashFunctionCtx &hashFunc) | |
| Scope: | class ara::crypto::cryp::PublicKey | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::cryp::PublicKey::HashPublicKey (HashFunctionCtx &hash`<br>`Func) const noexcept=0;` | |
| Parameters (in): | hashFunc | a hash-function instance that should be used the hashing |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | a buffer preallocated for the resulting hash value |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if size of the hash buffer is not enough for storing of the result |
| | CryptoErrorDomain::kIncompleteArg State | if the hashFunc context is not initialized |
| Header file: | #include "ara/crypto/cryp/cryobj/public_key.h" | |
| Description: | Calculate hash of the Public Key value. The original public key value BLOB is available via the Serializable interface. | |

⌋(*RS_CRYPTO_02202*)

## [SWS_CRYPT_22713]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | HashPublicKey(HashFunctionCtx &hashFunc) | |
| Scope: | class ara::crypto::cryp::PublicKey | |
| Syntax: | `template <typename Alloc = <implementation-defined>>`<br>`ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::Public`<br>`Key::HashPublicKey (HashFunctionCtx &hashFunc) const noexcept;` | |
| Template param: | Alloc | a custom allocator type of the output container |
| Parameters (in): | hashFunc | a hash-function instance that should be used the hashing |
| Return value: | ara::core::Result< ByteVector< Alloc > > | pre-reserved managed container for the resulting hash value |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if capacity of the hash buffer is not enough for storing of the result |
| | CryptoErrorDomain::kIncompleteArg State | if the hashFunc context is not initialized |
| Header file: | #include "ara/crypto/cryp/cryobj/public_key.h" | |
| Description: | Calculate hash of the Public Key value. This method sets the size of the output container according to actually saved value! The original public key value BLOB is available via the Serializable interface. | |

⌋(*RS_CRYPTO_02202*)

**[SWS_CRYPT_22914]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | AddEntropy(ReadOnlyMemRegion entropy) | |
| Scope: | class ara::crypto::cryp::RandomGeneratorCtx | |
| Syntax: | `virtual bool ara::crypto::cryp::RandomGeneratorCtx::AddEntropy (Read` `OnlyMemRegion entropy) noexcept=0;` | |
| Parameters (in): | entropy | a memory region with the additional entropy value |
| Return value: | bool | true if the method is supported and the entropy has been updated successfully |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/random_generator_ctx.h" | |
| Description: | Update the internal state of the RNG by mixing it with the provided additional entropy. This method is optional for implementation. An implementation of this method may "accumulate" provided entropy for future use. | |

⌋*(RS_CRYPTO_02206)*

**[SWS_CRYPT_22915]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Generate(std::uint32_t count) | |
| Scope: | class ara::crypto::cryp::RandomGeneratorCtx | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >` `ara::crypto::cryp::RandomGeneratorCtx::Generate (std::uint32_t count)` `noexcept=0;` | |
| Parameters (in): | count | number of random bytes to generate |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | a buffer filled with the generated random sequence |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if this context implements a local RNG (i.e., the RNG state is controlled by the application), and has to be seeded by the application because it either has not already been seeded or ran out of entropy. |
| | CryptoErrorDomain::kBusyResource | if this context implements a global RNG (i.e., the RNG state is controlled by the stack and not the application) that is currently out-of-entropy and therefore cannot provide the requested number of random bytes |
| Header file: | #include "ara/crypto/cryp/random_generator_ctx.h" | |
| Description: | Return an allocated buffer with a generated random sequence of the requested size. | |

⌋*(RS_CRYPTO_02206)*

**[SWS_CRYPT_22902]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetExtensionService() |
| Scope: | class ara::crypto::cryp::RandomGeneratorCtx |

▽

△

| Syntax: | `virtual ExtensionService::Uptr ara::crypto::cryp::RandomGenerator` `Ctx::GetExtensionService () const noexcept=0;` | |
|---|---|---|
| Return value: | ExtensionService::Uptr | – |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/random_generator_ctx.h" | |
| Description: | Get ExtensionService instance. | |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_22911]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Seed(ReadOnlyMemRegion seed) | |
| Scope: | class ara::crypto::cryp::RandomGeneratorCtx | |
| Syntax: | `virtual bool ara::crypto::cryp::RandomGeneratorCtx::Seed (ReadOnlyMem` `Region seed) noexcept=0;` | |
| Parameters (in): | seed | a memory region with the seed value |
| Return value: | bool | true if the method is supported and the state has been set successfully |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/random_generator_ctx.h" | |
| Description: | Set the internal state of the RNG using the provided seed. | |

⌋*(RS_CRYPTO_02206)*

## [SWS_CRYPT_22912]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Seed(const SecretSeed &seed) | |
| Scope: | class ara::crypto::cryp::RandomGeneratorCtx | |
| Syntax: | `virtual bool ara::crypto::cryp::RandomGeneratorCtx::Seed (const Secret` `Seed &seed) noexcept=0;` | |
| Parameters (in): | seed | a memory region with the seed value |
| Return value: | bool | true if the method is supported and the state has been set successfully |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/random_generator_ctx.h" | |
| Description: | Set the internal state of the RNG using the provided seed. | |

⌋*(RS_CRYPTO_02206)*

## [SWS_CRYPT_22913]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const SymmetricKey &key) | |
| Scope: | class ara::crypto::cryp::RandomGeneratorCtx | |

▽

△

| Syntax: | virtual bool ara::crypto::cryp::RandomGeneratorCtx::SetKey (const SymmetricKey &key) noexcept=0; | |
|---|---|---|
| Parameters (in): | key | a SymmetricKey with the key used as seed value |
| Return value: | bool | true if the method is supported and the key has been set successfully |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/random_generator_ctx.h" | |
| Description: | Set the internal state of the RNG using the provided seed. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02003*)

**[SWS_CRYPT_24811]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetAllowedUsage() | |
| Scope: | class ara::crypto::cryp::RestrictedUseObject | |
| Syntax: | virtual Usage ara::crypto::cryp::RestrictedUseObject::GetAllowedUsage () const noexcept=0; | |
| Return value: | Usage | a combination of bit-flags that specifies allowed applications of the object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/restricted_use_object.h" | |
| Description: | Get allowed usages of this object. | |

⌋(*RS_CRYPTO_02008*)

**[SWS_CRYPT_23011]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Clone(ReadOnlyMemRegion xorDelta=ReadOnlyMemRegion()) | |
| Scope: | class ara::crypto::cryp::SecretSeed | |
| Syntax: | virtual ara::core::Result<SecretSeed::Uptr> ara::crypto::cryp::Secret Seed::Clone (ReadOnlyMemRegion xorDelta=ReadOnlyMemRegion()) const noexcept=0; | |
| Parameters (in): | xorDelta | optional "delta" value that must be XOR-ed with the "cloned" copy of the original seed |
| Return value: | ara::core::Result< SecretSeed::Uptr > | unique smart pointer to "cloned" session Secret Seed object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" | |
| Description: | Clone this Secret Seed object to new session object. Created object instance is session and non-exportable, AllowedUsageFlags attribute of the "cloned" object is identical to this attribute of the source object! If size of the xorDelta argument is less than the value size of this seed then only correspondent number of leading bytes of the original seed should be XOR-ed, but the rest should be copied without change. If size of the xorDelta argument is larger than the value size of this seed then extra bytes of the xorDelta should be ignored. | |

⌋(*RS_CRYPTO_02007*)

## [SWS_CRYPT_23012]{OBSOLETE}

| Kind: | function |
|---|---|
| Symbol: | JumpFrom(const SecretSeed &from, std::int64_t steps) |
| Scope: | class ara::crypto::cryp::SecretSeed |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::SecretSeed::Jump From (const SecretSeed &from, std::int64_t steps) noexcept=0;` |
| Parameters (in): | from | source object that keeps the initial value for jumping from |
| | steps | number of steps for the "jump" |
| Return value: | ara::core::Result< void > | reference to this updated object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if this object and the from argument are associated with incompatible cryptographic algorithms |
| | CryptoErrorDomain::kInvalidInputSize | if value size of the from seed is less then value size of this one |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" | |
| Description: | Set value of this seed object as a "jump" from an initial state to specified number of steps, according to "counting" expression defined by a cryptographic algorithm associated with this object. steps may have positive and negative values that correspond to forward and backward direction of the "jump" respectively, but 0 value means only copy from value to this seed object. Seed size of the from argument always must be greater or equal of this seed size. | |

*(RS_CRYPTO_02007)*

## [SWS_CRYPT_23014]{OBSOLETE}

| Kind: | function |
|---|---|
| Symbol: | Jump(std::int64_t steps) |
| Scope: | class ara::crypto::cryp::SecretSeed |
| Syntax: | `virtual SecretSeed& ara::crypto::cryp::SecretSeed::Jump (std::int64_t steps) noexcept=0;` |
| Parameters (in): | steps | number of "steps" for jumping (forward or backward) from the current state |
| Return value: | SecretSeed & | reference to this updated object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" | |
| Description: | Set value of this seed object as a "jump" from it's current state to specified number of steps, according to "counting" expression defined by a cryptographic algorithm associated with this object. steps may have positive and negative values that correspond to forward and backward direction of the "jump" respectively, but 0 value means no changes of the current seed value. | |

*(RS_CRYPTO_02007)*

## [SWS_CRYPT_23013]{OBSOLETE}

| Kind: | function |
|---|---|
| Symbol: | Next() |
| Scope: | class ara::crypto::cryp::SecretSeed |
| Syntax: | `virtual SecretSeed& ara::crypto::cryp::SecretSeed::Next () noexcept=0;` |

▽

△

| Return value: | SecretSeed & | – |
|---|---|---|
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" | |
| Description: | Set next value of the secret seed according to "counting" expression defined by a cryptographic algorithm associated with this object. If the associated cryptographic algorithm doesn't specify a "counting" expression then generic increment operation must be implemented as default (little-endian notation, i.e. first byte is least significant). . | |

⌋(*RS_CRYPTO_02007*)

**[SWS_CRYPT_23015]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operatorˆ=(const SecretSeed &source) | |
| Scope: | class ara::crypto::cryp::SecretSeed | |
| Syntax: | `virtual SecretSeed& ara::crypto::cryp::SecretSeed::operatorˆ= (const SecretSeed &source) noexcept=0;` | |
| Parameters (in): | source | right argument for the XOR operation |
| Return value: | SecretSeed & | reference to this updated object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" | |
| Description: | XOR value of this seed object with another one and save result to this object. If seed sizes in this object and in the source argument are different then only correspondent number of leading bytes in this seed object should be updated. | |

⌋(*RS_CRYPTO_02007*)

**[SWS_CRYPT_23016]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operatorˆ=(ReadOnlyMemRegion source) | |
| Scope: | class ara::crypto::cryp::SecretSeed | |
| Syntax: | `virtual SecretSeed& ara::crypto::cryp::SecretSeed::operatorˆ= (Read OnlyMemRegion source) noexcept=0;` | |
| Parameters (in): | source | right argument for the XOR operation |
| Return value: | SecretSeed & | reference to this updated object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" | |
| Description: | XOR value of this seed object with provided memory region and save result to this object. If seed sizes in this object and in the source argument are different then only correspondent number of leading bytes of this seed object should be updated. | |

⌋(*RS_CRYPTO_02007*)

**[SWS_CRYPT_19906]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | CryptoException(ara::core::ErrorCode err) | |
| **Scope:** | class ara::crypto::CryptoException | |
| **Syntax:** | explicit ara::crypto::CryptoException::CryptoException (<br>ara::core::ErrorCode err) noexcept; | |
| **Parameters (in):** | err | the ErrorCode |
| **Exception Safety:** | noexcept | |
| **Header file:** | #include "ara/crypto/common/crypto_error_domain.h" | |
| **Description:** | Construct a new CryptoException from an ErrorCode. | |

⌋*(RS_CRYPTO_02310)*

**[SWS_CRYPT_23210]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | GetExtensionService() | |
| **Scope:** | class ara::crypto::cryp::SigEncodePrivateCtx | |
| **Syntax:** | virtual ExtensionService::Uptr ara::crypto::cryp::SigEncodePrivate<br>Ctx::GetExtensionService () const noexcept=0; | |
| **Return value:** | ExtensionService::Uptr | – |
| **Exception Safety:** | noexcept | |
| **Header file:** | #include "ara/crypto/cryp/sig_encode_private_ctx.h" | |
| **Description:** | Extension service member class. | |

⌋*(RS_CRYPTO_02006)*

**[SWS_CRYPT_23213]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | GetMaxInputSize(bool suppressPadding=false) | |
| **Scope:** | class ara::crypto::cryp::SigEncodePrivateCtx | |
| **Syntax:** | virtual std::size_t ara::crypto::cryp::SigEncodePrivateCtx::GetMax<br>InputSize (bool suppressPadding=false) const noexcept=0; | |
| **Parameters (in):** | suppressPadding | if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only |
| **Return value:** | std::size_t | maximum size of the input data block in bytes |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/cryp/sig_encode_private_ctx.h" | |
| **Description:** | Get maximum expected size of the input data block. If (IsEncryption() == false) then a value returned by this method is independent from the suppressPadding argument and it will be equal to the block size. | |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_23214]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetMaxOutputSize(bool suppressPadding=false) | |
| Scope: | class ara::crypto::cryp::SigEncodePrivateCtx | |
| Syntax: | `virtual std::size_t ara::crypto::cryp::SigEncodePrivateCtx::GetMax`<br>`OutputSize (bool suppressPadding=false) const noexcept=0;` | |
| Parameters (in): | suppressPadding | if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only |
| Return value: | std::size_t | maximum size of the output data block in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/sig_encode_private_ctx.h" | |
| Description: | Get maximum possible size of the output data block. If (IsEncryption() == true) then a value returned by this method is independent from the suppressPadding argument and will be equal to the block size. | |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_23215]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SignAndEncode(ReadOnlyMemRegion in) | |
| Scope: | class ara::crypto::cryp::SigEncodePrivateCtx | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::cryp::SigEncodePrivateCtx::SignAndEncode (ReadOnlyMem`<br>`Region in) const noexcept=0;` | |
| Parameters (in): | in | the input data block |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | the output buffer actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/sig_encode_private_ctx.h" | |
| Description: | Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutput Size(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppress Padding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! | |

⌋*(RS_CRYPTO_02202)*

### [SWS_CRYPT_23216]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SignAndEncode(ReadOnlyMemRegion in) | |
| Scope: | class ara::crypto::cryp::SigEncodePrivateCtx | |
| Syntax: | `template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::SigEncode PrivateCtx::SignAndEncode (ReadOnlyMemRegion in) const noexcept;` | |
| Template param: | Alloc | a custom allocator type of the output container |
| Parameters (in): | in | the input data block |
| Return value: | ara::core::Result< ByteVector< Alloc > > | the managed container for output block |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncorrectInput Size | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/sig_encode_private_ctx.h" | |
| Description: | Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! | |

⌋*(RS_CRYPTO_02202)*

### [SWS_CRYPT_23212]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Reset() | |
| Scope: | class ara::crypto::cryp::SigEncodePrivateCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::SigEncodePrivate Ctx::Reset () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/sig_encode_private_ctx.h" | |
| Description: | Clear the crypto context. | |

⌋*(RS_CRYPTO_02108)*

### [SWS_CRYPT_23211]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetKey(const PrivateKey &key) |
| Scope: | class ara::crypto::cryp::SigEncodePrivateCtx |

▽

△

| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::SigEncodePrivate Ctx::SetKey (const PrivateKey &key) noexcept=0; | |
|---|---|---|
| Parameters (in): | key | the source key object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/sig_encode_private_ctx.h" | |
| Description: | Set (deploy) a key to the sig encode private algorithm context. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02003*)

## [SWS_CRYPT_23311]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetHashAlgId() | |
| Scope: | class ara::crypto::cryp::Signature | |
| Syntax: | virtual CryptoPrimitiveId::AlgId ara::crypto::cryp::Signature::GetHash AlgId () const noexcept=0; | |
| Return value: | CryptoPrimitiveId::AlgId | ID of used hash algorithm only (without signature algorithm specification) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/cryobj/signature.h" | |
| Description: | Get an ID of hash algorithm used for this signature object production. | |

⌋(*RS_CRYPTO_02204*, *RS_CRYPTO_02203*, *RS_CRYPTO_02205*)

## [SWS_CRYPT_23312]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetRequiredHashSize() | |
| Scope: | class ara::crypto::cryp::Signature | |
| Syntax: | virtual std::size_t ara::crypto::cryp::Signature::GetRequiredHashSize () const noexcept=0; | |
| Return value: | std::size_t | required hash size in bytes |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/cryobj/signature.h" | |
| Description: | Get the hash size required by current signature algorithm. | |

⌋(*RS_CRYPTO_02309*)

### [SWS_CRYPT_29003]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetRequiredHashAlgId() | |
| Scope: | class ara::crypto::cryp::SignatureService | |
| Syntax: | virtual CryptoPrimitiveId::AlgId ara::crypto::cryp::Signature Service::GetRequiredHashAlgId () const noexcept=0; | |
| Return value: | CryptoPrimitiveId::AlgId | required hash algorithm ID or kAlgIdAny if the signature algorithm specification does not include a concrete hash function |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/signature_service.h" | |
| Description: | Get an ID of hash algorithm required by current signature algorithm. | |

⌋(RS_CRYPTO_02309)

### [SWS_CRYPT_29002]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetRequiredHashSize() | |
| Scope: | class ara::crypto::cryp::SignatureService | |
| Syntax: | virtual std::size_t ara::crypto::cryp::SignatureService::GetRequired HashSize () const noexcept=0; | |
| Return value: | std::size_t | required hash size in bytes |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/signature_service.h" | |
| Description: | Get the hash size required by current signature algorithm. | |

⌋(RS_CRYPTO_02309)

### [SWS_CRYPT_29004]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetSignatureSize() | |
| Scope: | class ara::crypto::cryp::SignatureService | |
| Syntax: | virtual std::size_t ara::crypto::cryp::SignatureService::GetSignature Size () const noexcept=0; | |
| Return value: | std::size_t | size of the signature value in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/signature_service.h" | |
| Description: | Get size of the signature value produced and required by the current algorithm. | |

⌋(RS_CRYPTO_02309)

## [SWS_CRYPT_23510]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetSignatureService() |
| Scope: | class ara::crypto::cryp::SignerPrivateCtx |
| Syntax: | virtual SignatureService::Uptr ara::crypto::cryp::SignerPrivate Ctx::GetSignatureService () const noexcept=0; |
| Return value: | SignatureService::Uptr | – |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/cryp/signer_private_ctx.h" |
| Description: | Get SignatureService instance. |

⌋(RS_CRYPTO_02006)

## [SWS_CRYPT_23516]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Reset() |
| Scope: | class ara::crypto::cryp::SignerPrivateCtx |
| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::SignerPrivate Ctx::Reset () noexcept=0; |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/signer_private_ctx.h" |
| Description: | Clear the crypto context. |

⌋(RS_CRYPTO_02108)

## [SWS_CRYPT_23515]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const PrivateKey &key) | |
| Scope: | class ara::crypto::cryp::SignerPrivateCtx | |
| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::SignerPrivate Ctx::SetKey (const PrivateKey &key) noexcept=0; | |
| Parameters (in): | key | the source key object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context is prohibited by /// the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/signer_private_ctx.h" | |
| Description: | Set (deploy) a key to the signer private algorithm context. | |

⌋(RS_CRYPTO_02001, RS_CRYPTO_02003)

**[SWS_CRYPT_23511]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | SignPreHashed(const HashFunctionCtx &hashFn, ReadOnlyMemRegion context=ReadOnlyMemRegion()) |
| **Scope:** | class ara::crypto::cryp::SignerPrivateCtx |
| **Syntax:** | `virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::Signer PrivateCtx::SignPreHashed (const HashFunctionCtx &hashFn, ReadOnlyMem Region context=ReadOnlyMemRegion()) const noexcept=0;` |
| **Parameters (in):** | hashFn | a finalized hash-function context that contains a digest value ready for sign |
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |
| **Return value:** | ara::core::Result< Signature::Uptrc > | unique smart pointer to serialized signature |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kInvalidArgument | if hash-function algorithm does not comply with the signature algorithm specification of this context |
| | CryptoErrorDomain::kInvalidInputSize | if the user supplied context has incorrect (or unsupported) size |
| | CryptoErrorDomain::kProcessingNot Finished | if the method hash.Finish() was not called before the call of this method |
| | CryptoErrorDomain::kUninitialized Context | this context was not initialized by a key value |
| **Header file:** | #include "ara/crypto/cryp/signer_private_ctx.h" | |
| **Description:** | Sign a provided digest value stored in the hash-function context. This method must put the hash-function algorithm ID and a COUID of the used key-pair to the resulting signature object! The user supplied context may be used for such algorithms as: Ed25519ctx, Ed25519ph, Ed448ph. If the target algorithm doesn't support the context argument then the empty (default) value must be supplied! | |

⌋(*RS_CRYPTO_02204*)

**[SWS_CRYPT_23512]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | Sign(ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) |
| **Scope:** | class ara::crypto::cryp::SignerPrivateCtx |
| **Syntax:** | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::SignerPrivateCtx::Sign (ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;` |
| **Parameters (in):** | value | the (pre-)hashed or direct message value that should be signed |
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::Byte > > | actual size of the signature value stored to the output buffer |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kInvalidInputSize | if size of the input value or context arguments are incorrect / unsupported |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| **Header file:** | #include "ara/crypto/cryp/signer_private_ctx.h" | |

▽

△

| Description: | Sign a directly provided hash or message value. This method can be used for implementation of the "multiple passes" signature algorithms that process a message directly, i.e. without "pre-hashing" (like Ed25519ctx). But also this method is suitable for implementation of the traditional signature schemes with pre-hashing (like Ed25519ph, Ed448ph, ECDSA). If the target algorithm doesn't support the context argument then the empty (default) value must be supplied! |
|---|---|

⌋*(RS_CRYPTO_02204)*

## [SWS_CRYPT_23513]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SignPreHashed(AlgId hashAlgId, ReadOnlyMemRegion hashValue, ReadOnlyMemRegion context=ReadOnlyMemRegion()) | |
| Scope: | class ara::crypto::cryp::SignerPrivateCtx | |
| Syntax: | `virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::Signer PrivateCtx::SignPreHashed (AlgId hashAlgId, ReadOnlyMemRegion hash Value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;` | |
| Parameters (in): | hashAlgId | hash function algorithm ID |
| | hashValue | hash function value (resulting digest without any truncations) |
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |
| Return value: | ara::core::Result< Signature::Uptrc > | unique smart pointer to serialized signature |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if hash-function algorithm does not comply with the signature algorithm specification of this context |
| | CryptoErrorDomain::kInvalidInputSize | if the user supplied context has incorrect (or unsupported) size |
| | CryptoErrorDomain::kUninitialized Context | this context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/signer_private_ctx.h" | |
| Description: | Sign a directly provided digest value and create the Signature object. This method must put the hash-function algorithm ID and a COUID of the used key-pair to the resulting signature object! The user supplied context may be used for such algorithms as: Ed25519ctx, Ed25519ph, Ed448ph. If the target algorithm doesn't support the context argument then the empty (default) value must be supplied! | |

⌋*(RS_CRYPTO_02204)*

## [SWS_CRYPT_23514]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Sign(ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) | |
| Scope: | class ara::crypto::cryp::SignerPrivateCtx | |
| Syntax: | `template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::SignerPrivate Ctx::Sign (ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnly MemRegion()) const noexcept;` | |
| Template param: | Alloc | a custom allocator type of the output container |

▽

△

| Parameters (in): | value | the (pre-)hashed or direct message value that should be signed |
|---|---|---|
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |
| Return value: | ara::core::Result< ByteVector< Alloc > > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if size of the input value or context arguments are incorrect / unsupported |
| | CryptoErrorDomain::kInsufficient Capacity | if capacity of the output signature container is not enough |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/signer_private_ctx.h" | |
| Description: | Sign a directly provided hash or message value. This method can be used for implementation of the "multiple passes" signature algorithms that process a message directly, i.e. without "pre-hashing" (like Ed25519ctx). But also this method is suitable for implementation of the traditional signature schemes with pre-hashing (like Ed25519ph, Ed448ph, ECDSA). This method sets the size of the output container according to actually saved value! If the target algorithm doesn't support the context argument then the empty (default) value must be supplied! | |

⌋(*RS_CRYPTO_02204*)

## [SWS_CRYPT_23620]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CountBytesInCache() |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `virtual std::size_t ara::crypto::cryp::StreamCipherCtx::CountBytesInCache () const noexcept=0;` |
| Return value: | std::size_t | number of bytes now kept in the context cache |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Count number of bytes now kept in the context cache. In block-wise modes if an application has supplied input data chunks with incomplete last block then the context saves the rest part of the last (incomplete) block to internal "cache" memory and wait a next call for additional input to complete this block. | |

⌋(*RS_CRYPTO_02302*)

## [SWS_CRYPT_23621]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | EstimateMaxInputSize(std::size_t outputCapacity) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `std::size_t ara::crypto::cryp::StreamCipherCtx::EstimateMaxInputSize (std::size_t outputCapacity) const noexcept;` |
| Parameters (in): | outputCapacity | capacity of the output buffer |
| Return value: | std::size_t | maximum number of input bytes |

▽

△

| Exception Safety: | noexcept |
|---|---|
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" |
| Description: | Estimate maximal number of input bytes that may be processed for filling of an output buffer without overflow. |

⌊*(RS_CRYPTO_02302)*

## [SWS_CRYPT_23622]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | EstimateRequiredCapacity(std::size_t inputSize, bool isFinal=false) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `std::size_t ara::crypto::cryp::StreamCipherCtx::EstimateRequired Capacity (std::size_t inputSize, bool isFinal=false) const noexcept;` |
| Parameters (in): | inputSize | size of input data |
| | isFinal | flag that indicates processing of the last data chunk (if true) |
| Return value: | std::size_t | required capacity of the output buffer (in bytes) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Estimate minimal required capacity of the output buffer, which is enough for saving a result of input data processing. | |

⌊*(RS_CRYPTO_02302)*

## [SWS_CRYPT_23618]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | FinishBytes(ReadOnlyMemRegion in) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::StreamCipherCtx::FinishBytes (ReadOnlyMemRegion in) noexcept=0;` |
| Parameters (in): | in | an input data buffer |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | an output data buffer |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if capacity of the output buffer is not enough |
| | CryptoErrorDomain::kInOutBuffers Intersect | if the input and output buffers intersect |
| | CryptoErrorDomain::kProcessingNot Started | if data processing was not started by a call of the Start() method |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |

▽

△

| Description: | Processe the final part of message (that may be not aligned to the block-size boundary). If (Is BytewiseMode() == false) then it must be: bs = GetBlockSize(), out.size() >= (((in.size() + bs * ((CryptoTransform::kEncrypt == GetTransformation().Value())) ? 2 : 1) |
|---|---|
| | 1) / bs) * bs) If (IsBytewiseMode() == true) then it must be: out.size() >= in.size() The input and output buffers must not intersect! Usage of this method is mandatory for processing of the last data chunk in block-wise modes! This method may be used for processing of a whole message in a single call (in any mode)! in an input data buffer an output data buffer CryptoError Domain::kInsufficientCapacity if capacity of the output buffer is not enough CryptoError Domain::kInOutBuffersIntersect if the input and output buffers intersect CryptoErrorDomain::k ProcessingNotStarted if data processing was not started by a call ofthe Start() method |

⌊*(RS_CRYPTO_02302)*

## **[SWS_CRYPT_23619]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | FinishBytes(ReadOnlyMemRegion in) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `template <typename Alloc = <implementation-defined>>`<br>`ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::StreamCipher`<br>`Ctx::FinishBytes (ReadOnlyMemRegion in) noexcept;` |
| Template param: | Alloc | a custom allocator type of the output container |
| Parameters (in): | in | an input data buffer The input buffer must not point inside the output container! |
| Return value: | ara::core::Result< ByteVector< Alloc > > | a managed container for output data |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if capacity of the output container is not enough |
| | CryptoErrorDomain::kInOutBuffers Intersect | if the input and output buffers intersect |
| | CryptoErrorDomain::kProcessingNot Started | if data processing was not started by a call of the Start() method |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Processe the final part of message (that may be not aligned to the block-size boundary). This method sets the size of the output container according to actually saved value. If (IsBytewise Mode() == false) then it must be: bs = GetBlockSize(), out.capacity() >= (((in.size() + bs * ((CryptoTransform::kEncrypt == GetTransformation.Value()) ? 2 : 1) - 1) / bs) * bs) If (Is BytewiseMode() == true) then it must be: out.capacity() >= in.size() Usage of this method is mandatory for processing of the last data chunk in block-wise modes! This method may be used for processing of a whole message in a single call (in any mode)! | |

⌊*(RS_CRYPTO_02302)*

## **[SWS_CRYPT_23602]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetBlockService() |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `virtual BlockService::Uptr ara::crypto::cryp::StreamCipherCtx::Get`<br>`BlockService () const noexcept=0;` |
| Return value: | BlockService::Uptr | – |

▽

$\triangle$

| Exception Safety: | noexcept |
|---|---|
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" |
| Description: | Get BlockService instance. |

⌋(*RS_CRYPTO_02006*)

## [SWS_CRYPT_23611]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsBytewiseMode() | |
| Scope: | class ara::crypto::cryp::StreamCipherCtx | |
| Syntax: | `virtual bool ara::crypto::cryp::StreamCipherCtx::IsBytewiseMode () const noexcept=0;` | |
| Return value: | bool | true if the mode can process messages the byte-by-byte (without padding up to the block boundary) and false if only the block-by-block (only full blocks can be processed, the padding is mandatory) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Check the operation mode for the bytewise property. | |

⌋(*RS_CRYPTO_02309*)

## [SWS_CRYPT_23624]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetTransformation() | |
| Scope: | class ara::crypto::cryp::StreamCipherCtx | |
| Syntax: | `virtual ara::core::Result<CryptoTransform> ara::crypto::cryp::StreamCipherCtx::GetTransformation () const noexcept=0;` | |
| Return value: | ara::core::Result< CryptoTransform > | CryptoTransform |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Get the kind of transformation configured for this context: kEncrypt or kDecrypt. | |

⌋(*RS_CRYPTO_02309*)

## [SWS_CRYPT_23612]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IsSeekableMode() |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |

$\triangledown$

△

| Syntax: | virtual bool ara::crypto::cryp::StreamCipherCtx::IsSeekableMode () const noexcept=0; | |
|---|---|---|
| Return value: | bool | true the seek operation is supported in the current mode and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Check if the seek operation is supported in the current mode. | |

⌋(*RS_CRYPTO_02309*)

## [SWS_CRYPT_23614]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ProcessBlocks(ReadOnlyMemRegion in) | |
| Scope: | class ara::crypto::cryp::StreamCipherCtx | |
| Syntax: | virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::StreamCipherCtx::ProcessBlocks (ReadOnlyMemRegion in) noexcept=0; | |
| Parameters (in): | in | an input data buffer |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | an output data buffer |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Arguments | if sizes of the input and output buffers are not equal |
| | CryptoErrorDomain::kInvalidInputSize | if size of the input buffer is not divisible by the block size (see GetBlockSize()) |
| | CryptoErrorDomain::kInOutBuffers Intersect | if the input and output buffers partially intersect |
| | CryptoErrorDomain::kInvalidUsage Order | if this method is called after processing of non-aligned data (to the block-size boundary) |
| | CryptoErrorDomain::kProcessingNot Started | if the data processing was not started by a call of the Start() method |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Processe initial parts of message aligned to the block-size boundary. It is a copy-optimized method that doesn't use the internal cache buffer! It can be used only before processing of any non-aligned to the block-size boundary data. Pointers to the input and output buffers must be aligned to the block-size boundary! The input and output buffers may completely coincide, but they must not partially intersect! | |

⌋(*RS_CRYPTO_02302*)

## [SWS_CRYPT_23615]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ProcessBlocks(ReadWriteMemRegion inOut) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | virtual ara::core::Result<void> ara::crypto::cryp::StreamCipher Ctx::ProcessBlocks (ReadWriteMemRegion inOut) noexcept=0; |

▽

△

| Parameters (inout): | inOut | an input and output data buffer, i.e. the whole buffer should be updated |
|---|---|---|
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if size of the inOut buffer is not divisible by the block size (see GetBlockSize()) |
| | CryptoErrorDomain::kInvalidUsage Order | if this method is called after processing of non-aligned data (to the block-size boundary) |
| | CryptoErrorDomain::kProcessingNot Started | if the data processing was not started by a call of the Start() method |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Processe initial parts of message aligned to the block-size boundary. It is a copy-optimized method that doesn't use internal cache buffer! It can be used up to first non-block aligned data processing. Pointer to the input-output buffer must be aligned to the block-size boundary! | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_23616]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ProcessBytes(ReadOnlyMemRegion in) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | ```virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::StreamCipherCtx::ProcessBytes (ReadOnlyMemRegion in) noexcept=0;``` |
| Parameters (in): | in | an input data buffer |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | an output data buffer |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if the output buffer has capacity insufficient for placing of the transformation result |
| | CryptoErrorDomain::kInOutBuffers Intersect | if the input and output buffers intersect |
| | CryptoErrorDomain::kProcessingNot Started | if data processing was not started by a call of the Start() method |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Processe a non-final part of message (that is not aligned to the block-size boundary). If (Is BytewiseMode() == false) then it must be: bs = GetBlockSize(), out.size() >= (((in.size() + bs - 1) / bs) * bs) If (IsBytewiseMode() == true) then it must be: out.size() >= in.size() The input and output buffers must not intersect! This method is "copy inefficient", therefore it should be used only in conditions when an application cannot control the chunking of the original message! | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_23617]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ProcessBytes(ReadOnlyMemRegion in) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |

▽

△

| Syntax: | `template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::StreamCipherCtx::ProcessBytes (ReadOnlyMemRegion in) noexcept;` | |
|---|---|---|
| Template param: | Alloc | a custom allocator type of the output container |
| Parameters (in): | in | an input data buffer |
| Return value: | ara::core::Result< ByteVector< Alloc > > | a managed container for the output data |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if capacity of the output container is not enough |
| | CryptoErrorDomain::kInOutBuffers Intersect | if the input buffer points inside of the preallocated output container |
| | : | CryptoErrorDomain::kProcessingNotStarted if data processing was not started by a call of the Start() method |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Processes a non-final part of message (that is not aligned to the block-size boundary). This method sets size of the output container according to actually saved value. If (IsBytewiseMode() == false) then it must be: bs = GetBlockSize(), out.capacity() >= (((in.size() + bs - 1) / bs) * bs) If (IsBytewiseMode() == true) then it must be: out.capacity() >= in.size() This method is "copy inefficient", therefore it should be used only in conditions when an application cannot control the chunking of the original message! The input buffer must not point inside the output container! | |

⌋*(RS_CRYPTO_02302)*

### [SWS_CRYPT_23627]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Reset() |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::Reset () noexcept=0;` |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" |
| Description: | Clear the crypto context. |

⌋*(RS_CRYPTO_02108)*

### [SWS_CRYPT_23613]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Seek(std::int64_t offset, bool fromBegin=true) | |
| Scope: | class ara::crypto::cryp::StreamCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::Seek (std::int64_t offset, bool fromBegin=true) noexcept=0;` | |
| Parameters (in): | offset | the offset value in bytes, relative to begin or current position in the gamma stream |
| | fromBegin | the starting point for positioning within the stream: from begin (if true) or from current position (if false) |

▽

$\triangle$

| Return value: | ara::core::Result< void > | – |
|---|---|---|
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnsupported | if the seek operation is not supported by the current mode |
| | CryptoErrorDomain::kProcessingNotStarted | if the data processing was not started by a call of the Start() method |
| | CryptoErrorDomain::kBelowBoundary | if the offset value is incorrect (in context of the the fromBegin argument), i.e. it points before begin of the stream (note: it is an optional error condition) |
| | CryptoErrorDomain::kInvalidArgument | if the offset is not aligned to the required boundary (see IsBytewiseMode()) |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Set the position of the next byte within the stream of the encryption/decryption gamma. | |

⌋*(RS_CRYPTO_02304)*

**[SWS_CRYPT_23623]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::SetKey (const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt) noexcept=0;` |

| Parameters (in): | key | the source key object |
|---|---|---|
| | transform | the "direction" indicator: deploy the key for direct transformation (if true) or for reverse one (if false) |

| Return value: | ara::core::Result< void > | – |
|---|---|---|
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatibleObject | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Set (deploy) a key to the stream chiper algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

**[SWS_CRYPT_23625]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Start(ReadOnlyMemRegion iv=ReadOnlyMemRegion()) |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::Start (ReadOnlyMemRegion iv=ReadOnlyMemRegion()) noexcept=0;` |
| Parameters (in): | iv | an optional Initialization Vector (IV) or "nonce" value |

$\triangledown$

△

| Return value: | ara::core::Result< void > | – |
|---|---|---|
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by deploying a key |
| | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false) |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_23626]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Start(const SecretSeed &iv) | |
| Scope: | class ara::crypto::cryp::StreamCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::Start (const SecretSeed &iv) noexcept=0;` | |
| Parameters (in): | iv | the Initialization Vector (IV) or "nonce" object |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by deploying a key |
| | CryptoErrorDomain::kInvalidInputSize | if the size of provided IV is not supported (i.e. if it is not enough for the initialization) |
| | CryptoErrorDomain::kUnsupported | if the base algorithm (or its current implementation) principally doesn't support the IV variation |
| | CryptoErrorDomain::kUsageViolation | if this transformation type is prohibited by the "allowed usage" restrictions of the provided Secret Seed object |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" | |
| Description: | Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence. | |

⌋*(RS_CRYPTO_02302)*

## [SWS_CRYPT_23702]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetCryptoService() |
| Scope: | class ara::crypto::cryp::SymmetricBlockCipherCtx |
| Syntax: | `virtual CryptoService::Uptr ara::crypto::cryp::SymmetricBlockCipherCtx::GetCryptoService () const noexcept=0;` |

▽

△

| Return value: | CryptoService::Uptr | – |
|---|---|---|
| **Exception Safety:** | noexcept | |
| **Header file:** | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" | |
| **Description:** | Get CryptoService instance. | |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_23711]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetTransformation() |
| **Scope:** | class ara::crypto::cryp::SymmetricBlockCipherCtx |
| **Syntax:** | `virtual ara::core::Result<CryptoTransform> ara::crypto::cryp::SymmetricBlockCipherCtx::GetTransformation () const noexcept=0;` |

| Return value: | ara::core::Result< CryptoTransform > | CryptoTransform |
|---|---|---|
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUninitialized Context, | if SetKey() has not been called yet. |
| **Header file:** | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" | |
| **Description:** | Get the kind of transformation configured for this context: kEncrypt or kDecrypt. | |

⌋*(RS_CRYPTO_02309)*

## [SWS_CRYPT_23716]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false) |
| **Scope:** | class ara::crypto::cryp::SymmetricBlockCipherCtx |
| **Syntax:** | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlock (ReadOnlyMem Region in, bool suppressPadding=false) const noexcept=0;` |

| Parameters (in): | in | the input data block |
|---|---|---|
| | suppressPadding | if true then this method doesn't apply padding, hence the input buffer be of the same size as the block-size, i.e. either the data to be processed exactly fits the block-size or the user must apply padding to the same effect. |
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::Byte > > | the output buffer containing the transformation result |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | : | CryptoErrorDomain::kInvalidInputSize if the boolean parameter \ARApiRef{suppressPadding} was set to TRUE and the provided input buffer does not match the block-size. |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by calling SetKey() |
| **Header file:** | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" | |
| **Description:** | Process (encrypt / decrypt) an input block according to the configuration. | |

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_23717]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false) | |
| **Scope:** | class ara::crypto::cryp::SymmetricBlockCipherCtx | |
| **Syntax:** | `template <typename Alloc = <implementation-defined>>`<br>`ara::core::Result<ByteVector<Alloc> > ara::crypto::cryp::Symmetric`<br>`BlockCipherCtx::ProcessBlock (ReadOnlyMemRegion in, bool suppress`<br>`Padding=false) const noexcept;` | |
| **Template param:** | Alloc | a custom allocator type of the output container |
| **Parameters (in):** | in | the input data block |
| | suppressPadding | if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data |
| **Return value:** | ara::core::Result< ByteVector< Alloc > > | the managed container for output block |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kIncorrectInputSize | if the mentioned above rules about the input size is violated |
| | CryptoErrorDomain::kInsufficientCapacity | if the out.size() is not enough to store the transformation result |
| | CryptoErrorDomain::kUninitializedContext | if the context was not initialized by a key value |
| **Header file:** | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" | |
| **Description:** | Process (encrypt / decrypt) an input block according to the configuration. | |

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_23715]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | ProcessBlocks(ReadOnlyMemRegion in) | |
| **Scope:** | class ara::crypto::cryp::SymmetricBlockCipherCtx | |
| **Syntax:** | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlocks (ReadOnlyMem`<br>`Region in) const noexcept=0;` | |
| **Parameters (in):** | in | an input data buffer |
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::Byte > > | an output data buffer |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUninitializedContext | if the context was not initialized by a key value |
| | CryptoErrorDomain::kInvalidInputSize | if size of the input buffer is not divisible by the block size (see GetBlockSize()) |
| **Header file:** | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" | |
| **Description:** | Process (encrypt / decrypt) an input block according to the configuration. The in must have a size that is divisible by the block size (see GetBlockSize()). The pointer to the input buffer must be aligned to the block-size boundary! | |

⌋*(RS_CRYPTO_02302)*

### [SWS_CRYPT_23712]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Reset() | |
| Scope: | class ara::crypto::cryp::SymmetricBlockCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::SymmetricBlock`<br>`CipherCtx::Reset () noexcept=0;` | |
| Return value: | ara::core::Result< void > | true if the transformation requires the maximum size of input data and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUninitialized Context | if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet |
| Header file: | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" | |
| Description: | Indicate that the currently configured transformation accepts only complete blocks of input data.<br><br>Clear the crypto context. | |

⌋*(RS_CRYPTO_02309, RS_CRYPTO_02108)*

### [SWS_CRYPT_23710]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt) | |
| Scope: | class ara::crypto::cryp::SymmetricBlockCipherCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::SymmetricBlock`<br>`CipherCtx::SetKey (const SymmetricKey &key, CryptoTransform`<br>`transform=CryptoTransform::kEncrypt) noexcept=0;` | |
| Parameters (in): | key | the source key object |
| | transform | the "direction" indicator: deploy the key for direct transformation (if true) or for reverse one (if false) |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object belongs to a different CryptoProvider instance |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" | |
| Description: | Set (deploy) a key to the symmetric algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

### [SWS_CRYPT_24013]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CalculateWrappedKeySize(std::size_t keyLength) |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Syntax: | `virtual std::size_t ara::crypto::cryp::SymmetricKeyWrapper`<br>`Ctx::CalculateWrappedKeySize (std::size_t keyLength) const noexcept=0;` |

▽

△

| Parameters (in): | keyLength | original key length in bits |
|---|---|---|
| Return value: | std::size_t | size of the wrapped key in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |
| Description: | Calculate size of the wrapped key in bytes from original key length in bits. This method can be useful for some implementations different from RFC3394 / RFC5649. | |

⌋*(RS_CRYPTO_02201)*

## [SWS_CRYPT_24002]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetExtensionService() |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Syntax: | `virtual ExtensionService::Uptr ara::crypto::cryp::SymmetricKeyWrapperCtx::GetExtensionService () const noexcept=0;` |
| Return value: | ExtensionService::Uptr | – |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |
| Description: | Get ExtensionService instance. | |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_24012]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetMaxTargetKeyLength() |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Syntax: | `virtual std::size_t ara::crypto::cryp::SymmetricKeyWrapperCtx::GetMax TargetKeyLength () const noexcept=0;` |
| Return value: | std::size_t | maximum length of the target key in bits |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |
| Description: | Get maximum length of the target key supported by the implementation. This method can be useful for some implementations different from RFC3394 / RFC5649. | |

⌋*(RS_CRYPTO_02201)*

## [SWS_CRYPT_24011]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetTargetKeyGranularity() |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Syntax: | `virtual std::size_t ara::crypto::cryp::SymmetricKeyWrapperCtx::Get TargetKeyGranularity () const noexcept=0;` |
| Return value: | std::size_t | size of the block in bytes |
| Exception Safety: | noexcept | |

▽

△

| Thread Safety: | Thread-safe |
|---|---|
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" |
| Description: | Get expected granularity of the target key (block size). If the class implements RFC3394 (KW without padding) then this method should return 8 (i.e. 8 octets = 64 bits). If the class implements RFC5649 (KW with padding) then this method should return 1 (i.e. 1 octet = 8 bits). |

⌋*(RS_CRYPTO_02201)*

## [SWS_CRYPT_24019]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Reset() |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::SymmetricKeyWrapperCtx::Reset () noexcept=0;` |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" |
| Description: | Clear the crypto context. |

⌋*(RS_CRYPTO_02108)*

## [SWS_CRYPT_24018]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const SymmetricKey &key, CryptoTransform transform) | |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::SymmetricKeyWrapperCtx::SetKey (const SymmetricKey &key, CryptoTransform transform) noexcept=0;` | |
| Parameters (in): | key | the source key object |
| | transform | the "direction" indicator: deploy the key for direct transformation (if true) or for reverse one (if false) |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrorDomain::kUsageViolation | if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |
| Description: | Set (deploy) a key to the symmetric key wrapper algorithm context. | |

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

### [SWS_CRYPT_24017]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | UnwrapConcreteKey(ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage) | |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx | |
| Syntax: | `template <typename ExpectedKey>`<br>`ara::core::Result<typename ExpectedKey::Uptrc>`<br>`ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapConcreteKey (ReadOnly`<br>`MemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage)`<br>`noexcept;` | |
| Template param: | ExpectedKey | the expected type of concrete key |
| Parameters (in): | wrappedKey | a memory region that contains wrapped key |
| | algId | an identifier of the target symmetric crypto algorithm |
| | allowedUsage | bit-flags that define a list of allowed transformations' types in which the target key can be used |
| Return value: | ara::core::Result< typename Expected Key::Uptrc > | unique smart pointer to ExpectedKey object, which keeps unwrapped key material |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if the size of provided wrapped key is unsupported |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |
| Description: | Execute the "key unwrap" operation for provided BLOB and produce a Key object of expected type. For additional details see UnwrapKey() | |

⌋*(RS_CRYPTO_02115)*

### [SWS_CRYPT_24016]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | UnwrapKey(ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage) | |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx | |
| Syntax: | `virtual ara::core::Result<RestrictedUseObject::Uptrc>`<br>`ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapKey (ReadOnlyMem`<br>`Region wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage) const`<br>`noexcept=0;` | |
| Parameters (in): | wrappedKey | a memory region that contains wrapped key |
| | algId | an identifier of the target symmetric crypto algorithm |
| | allowedUsage | bit-flags that define a list of allowed transformations' types in which the target key can be used |
| Return value: | ara::core::Result< RestrictedUse Object::Uptrc > | unique smart pointer to Key object, which keeps unwrapped key material |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if the size of provided wrapped key is unsupported |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |

▽

△

| Description: | Execute the "key unwrap" operation for provided BLOB and produce Key object. This method should be compliant to RFC3394 or RFC5649, if implementation is based on the AES block cipher and applied to an AES key. The created Key object has following attributes: session and non-exportable (because it was imported without meta-information)! SymmetricKey may be unwrapped in following way: SymmetricKey::Uptrc key = SymmetricKey::Cast(Unwrap Key(wrappedKey, ...)); PrivateKey may be unwrapped in following way: PrivateKey::Uptrc key = PrivateKey::Cast(UnwrapKey(wrappedKey, ...)); In both examples the Cast() method may additionally throw the BadObjectTypeException if an actual type of the unwrapped key differs from the target one! |
|---|---|

⌋*(RS_CRYPTO_02115)*

## [SWS_CRYPT_24015]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | UnwrapSeed(ReadOnlyMemRegion wrappedSeed, AlgId targetAlgId, SecretSeed::Usage allowedUsage) |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Syntax: | `virtual ara::core::Result<SecretSeed::Uptrc>`<br>`ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapSeed (ReadOnlyMem`<br>`Region wrappedSeed, AlgId targetAlgId, SecretSeed::Usage allowedUsage)`<br>`const noexcept=0;` |
| Parameters (in): | wrappedSeed | a memory region that contains wrapped seed |
| | targetAlgId | the target symmetric algorithm identifier (also defines a target seed-length) |
| | allowedUsage | allowed usage scope of the target seed |
| Return value: | ara::core::Result< SecretSeed::Uptrc > | unique smart pointer to SecretSeed object, which keeps unwrapped key material |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidInputSize | if the size of provided wrapped seed is unsupported |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |
| Description: | Execute the "key unwrap" operation for provided BLOB and produce SecretSeed object. This method should be compliant to RFC3394 or RFC5649, if implementation is based on the AES block cipher and applied to an AES key material. The created SecretSeed object has following attributes: session and non-exportable (because it was imported without meta-information). |

⌋*(RS_CRYPTO_02007)*

## [SWS_CRYPT_24014]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | WrapKeyMaterial(const RestrictedUseObject &key) |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::cryp::SymmetricKeyWrapperCtx::WrapKeyMaterial (const`<br>`RestrictedUseObject &key) const noexcept=0;` |
| Parameters (in): | key | a key that should be wrapped |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | – |
| Exception Safety: | noexcept | |

▽

△

| Thread Safety: | Thread-safe | |
|---|---|---|
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if the size of the wrapped buffer is not enough for storing the result |
| | CryptoErrorDomain::kInvalidInputSize | if the key object has an unsupported length |
| | CryptoErrorDomain::kUninitialized Context | if the context was not initialized by a key value |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" | |
| Description: | Execute the "key wrap" operation for the provided key material. This method should be compliant to RFC3394 or RFC5649, if an implementation is based on the AES block cipher and applied to an AES key. Method CalculateWrappedKeySize() can be used for size calculation of the required output buffer. | |

⌋*(RS_CRYPTO_02201)*

## [SWS_CRYPT_24102]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetSignatureService() | |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx | |
| Syntax: | `virtual SignatureService::Uptr ara::crypto::cryp::VerifierPublic Ctx::GetSignatureService () const noexcept=0;` | |
| Return value: | SignatureService::Uptr | – |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" | |
| Description: | Extension service member class. | |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_24116]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Reset() | |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::VerifierPublic Ctx::Reset () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" | |
| Description: | Clear the crypto context. | |

⌋*(RS_CRYPTO_02108)*

## [SWS_CRYPT_24115]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetKey(const PublicKey &key) | |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::cryp::VerifierPublic Ctx::SetKey (const PublicKey &key) noexcept=0;` | |

▽

△

| Parameters (in): | key | the source key object |
|---|---|---|
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kIncompatibleObject | if the provided key object is incompatible with this symmetric key context |
| | CryptoErrc::kUsageViolation | if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" | |
| Description: | Set (deploy) a key to the verifier public algorithm context. | |

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02003*)

## [SWS_CRYPT_24111]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | VerifyPrehashed(CryptoAlgId hashAlgId, ReadOnlyMemRegion hashValue, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) | |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx | |
| Syntax: | `virtual ara::core::Result<bool> ara::crypto::cryp::VerifierPublic Ctx::VerifyPrehashed (CryptoAlgId hashAlgId, ReadOnlyMemRegion hash Value, const Signature &signature, ReadOnlyMemRegion context=ReadOnly MemRegion()) const noexcept=0;` | |
| Parameters (in): | hashAlgId | hash function algorithm ID |
| | hashValue | hash function value (resulting digest without any truncations) |
| | signature | the signature object for verification |
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |
| Return value: | ara::core::Result< bool > | true if the signature was verified successfully and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kProcessingNotFinished | if the method hashFn.Finish() was not called before this method call |
| | CryptoErrc::kInvalidArgument | if the CryptoAlgId of hashFn differs from the Crypto AlgId of this context |
| | CryptoErrc::kInvalidInputSize | if the size of the supplied context is incompatible with the configured signature algorithm. |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" | |
| Description: | Verify signature by a digest value stored in the hash-function context. This is a pass-through interface to SWS_CRYPT_24113 for developer convenience, i.e. it adds additional input checks amd then calls the verify() interface from SWS_CRYPT_24113. | |

⌋(*RS_CRYPTO_02204*)

## [SWS_CRYPT_24112]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Verify(ReadOnlyMemRegion value, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx |
| Syntax: | `virtual ara::core::Result<bool> ara::crypto::cryp::VerifierPublic Ctx::Verify (ReadOnlyMemRegion value, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;` |
| Parameters (in): | value | the (pre-)hashed or direct message value that should be verified |
| | signature | the signature BLOB for the verification (the BLOB contains a plain sequence of the digital signature components located in fixed/maximum length fields defined by the algorithm specification, and each component is presented by a raw bytes sequence padded by zeroes to full length of the field; e.g. in case of (EC)DSA-256 (i.e. length of the q module is 256 bits) the signature BLOB must have two fixed-size fields: 32 + 32 bytes, for R and S components respectively, i.e. total BLOB size is 64 bytes) |
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |
| Return value: | ara::core::Result< bool > | true if the signature was verified successfully and false otherwise |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Errors: | CryptoErrc::kUninitializedContext | if the context was not initialized by a key value |
| | CryptoErrc::kInvalidInputSize | if the context argument has unsupported size |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" |
| Description: | Verify signature BLOB by a directly provided hash or message value. This method can be used for implementation of the "multiple passes" signature algorithms that process a message directly, i.e. without "pre-hashing" (like Ed25519ctx). But also this method is suitable for implementation of the traditional signature schemes with pre-hashing (like Ed25519ph, Ed448ph, ECDSA). If the target algorithm doesn't support the context argument then the empty (default) value must be supplied! The user supplied context may be used for such algorithms as: Ed25519ctx, Ed25519ph, Ed448ph. |

⌋(*RS_CRYPTO_02204*)

## [SWS_CRYPT_24113]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | VerifyPrehashed(const HashFunctionCtx &hashFn, const Signature &signature, ReadOnlyMem Region context=ReadOnlyMemRegion()) |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx |
| Syntax: | `virtual ara::core::Result<bool> ara::crypto::cryp::VerifierPublic Ctx::VerifyPrehashed (const HashFunctionCtx &hashFn, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;` |
| Parameters (in): | hashFn | hash function to be used for hashing |
| | signature | the signature object for the verification |
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |

▽

$\triangle$

| Return value: | ara::core::Result< bool > | true if the signature was verified successfully and false otherwise |
|---|---|---|
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kIncompatibleObject | if the CryptoAlgId of this context does not match the CryptoAlgId of signature; or the required CryptoAlgId of the hash is not kAlgIdDefault and the required hash CryptoAlgId of this context does not match hashAlgId or the hash CryptoAlgId of signature |
| | CryptoErrc::kIncompatibleArguments | if the provided hashAlgId is not kAlgIdDefault and the CryptoAlgId of the provided signature object does not match the provided hashAlgId |
| | CryptoErrc::kBadObjectReference | if the provided signature object does not reference the public key loaded to the context, i.e. if the COUID of the public key in the context is not equal to the COUID referenced from the signature object. |
| | CryptoErrc::kInvalidInputSize | if the size of the supplied context or hashValue is incompatible with the configured signature algorithm. |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" | |
| Description: | Verify signature by a digest value stored in the hash-function context. This is a pass-through interface to SWS_CRYPT_24112 for developer convenience, i.e. it adds additional input checks amd then calls the default verify() interface. | |

⌋*(RS_CRYPTO_02204)*

**[SWS_CRYPT_24114]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | VerifyPrehashed(const HashFunctionCtx &hashFn, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx |
| Syntax: | `virtual ara::core::Result<bool> ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed (const HashFunctionCtx &hashFn, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;` |

| Parameters (in): | hashFn | hash function to be used for hashing |
|---|---|---|
| | signature | the data BLOB to be verified |
| | context | an optional user supplied "context" (its support depends from concrete algorithm) |
| Return value: | ara::core::Result< bool > | true if the signature was verified successfully and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrc::kProcessingNotFinished | if the method hashFn.Finish() was not called before this method call |
| | CryptoErrc::kInvalidArgument | if the CryptoAlgId of hashFn differs from the CryptoAlgId of this context |
| | CryptoErrc::kInvalidInputSize | if the size of the supplied context or signature is incompatible with the configured signature algorithm. |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" | |

$\triangledown$

△

| Description: | Verify signature by a digest value stored in the hash-function context. This is a pass-through interface to SWS_CRYPT_24112 for developer convenience, i.e. it adds additional input checks amd then calls the default verify() interface. |
|---|---|

⌋*(RS_CRYPTO_02204)*

## [SWS_CRYPT_24101]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::VerifierPublicCtx |
| Derived from: | std::unique_ptr<VerifierPublicCtx> |
| Syntax: | `using ara::crypto::cryp::VerifierPublicCtx::Uptr = std::unique_ptr<VerifierPublicCtx>;` |
| Header file: | #include "ara/crypto/cryp/verifier_public_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02204)*

## [SWS_CRYPT_20101]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::AuthCipherCtx |
| Derived from: | std::unique_ptr<AuthCipherCtx> |
| Syntax: | `using ara::crypto::cryp::AuthCipherCtx::Uptr = std::unique_ptr<AuthCipherCtx>;` |
| Header file: | #include "ara/crypto/cryp/auth_cipher_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02207)*

## [SWS_CRYPT_24802]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::RestrictedUseObject |
| Derived from: | std::unique_ptr<const RestrictedUseObject> |
| Syntax: | `using ara::crypto::cryp::RestrictedUseObject::Uptrc = std::unique_ptr<const RestrictedUseObject>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/restricted_use_object.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02403)*

## [SWS_CRYPT_29031]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::BlockService |
| Derived from: | std::unique_ptr<BlockService> |
| Syntax: | using ara::crypto::cryp::BlockService::Uptr = std::unique_ptr<Block Service>; |
| Header file: | #include "ara/crypto/cryp/block_service.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02309)*

## [SWS_CRYPT_20402]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | AlgId |
| Scope: | class ara::crypto::cryp::CryptoContext |
| Derived from: | CryptoAlgId |
| Syntax: | using ara::crypto::cryp::CryptoContext::AlgId = CryptoAlgId; |
| Header file: | #include "ara/crypto/cryp/crypto_context.h" |
| Description: | Type definition of vendor specific binary Crypto Primitive ID. |

⌋*(RS_CRYPTO_02008)*

## [SWS_CRYPT_20504]{DRAFT} ⌈

| Kind: | struct |
|---|---|
| Symbol: | COIdentifier |
| Scope: | class ara::crypto::cryp::CryptoObject |
| Syntax: | struct ara::crypto::cryp::CryptoObject::COIdentifier {...}; |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | Unique identifier of this CryptoObject. |

⌋*(RS_CRYPTO_02005)*

## [SWS_CRYPT_20502]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::CryptoObject |
| Derived from: | std::unique_ptr<const CryptoObject> |
| Syntax: | using ara::crypto::cryp::CryptoObject::Uptrc = std::unique_ptr<const CryptoObject>; |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | Unique smart pointer of the constant interface. |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_20501]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::CryptoObject |
| Derived from: | std::unique_ptr<CryptoObject> |
| Syntax: | `using ara::crypto::cryp::CryptoObject::Uptr = std::unique_ptr<Crypto Object>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_20641]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | AlgId |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId |
| Derived from: | CryptoAlgId |
| Syntax: | `using ara::crypto::cryp::CryptoPrimitiveId::AlgId = CryptoAlgId;` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" |
| Description: | Type definition of vendor specific binary Crypto Primitive ID. |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_20644]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId |
| Derived from: | std::unique_ptr<const CryptoPrimitiveId> |
| Syntax: | `using ara::crypto::cryp::CryptoPrimitiveId::Uptrc = std::unique_ptr<const CryptoPrimitiveId>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" |
| Description: | type definition pointer |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_20643]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::CryptoPrimitiveId |
| Derived from: | std::unique_ptr<CryptoPrimitiveId> |
| Syntax: | `using ara::crypto::cryp::CryptoPrimitiveId::Uptr = std::unique_ptr<CryptoPrimitiveId>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_primitive_id.h" |
| Description: | type definition pointer to const |

⌋*(RS_CRYPTO_02005)*

## [SWS_CRYPT_20703]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | AlgId |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Derived from: | CryptoPrimitiveId::AlgId |
| Syntax: | `using ara::crypto::cryp::CryptoProvider::AlgId = CryptoPrimitiveId::AlgId;` |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" |
| Description: | A short alias for Algorithm ID type definition. |

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

## [SWS_CRYPT_20701]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::CryptoProvider |
| Derived from: | std::unique_ptr<CryptoProvider> |
| Syntax: | `using ara::crypto::cryp::CryptoProvider::Uptr = std::unique_ptr<CryptoProvider>;` |
| Header file: | #include "ara/crypto/cryp/crypto_provider.h" |
| Description: | Shared smart pointer of the interface. |

⌋*(RS_CRYPTO_02109)*

## [SWS_CRYPT_29024]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::CryptoService |
| Derived from: | std::unique_ptr<CryptoService> |
| Syntax: | `using ara::crypto::cryp::CryptoService::Uptr = std::unique_ptr<CryptoService>;` |
| Header file: | #include "ara/crypto/cryp/crypto_service.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02309)*

## [SWS_CRYPT_20801]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::DecryptorPrivateCtx |
| Derived from: | std::unique_ptr<DecryptorPrivateCtx> |
| Syntax: | `using ara::crypto::cryp::DecryptorPrivateCtx::Uptr = std::unique_ptr<DecryptorPrivateCtx>;` |
| Header file: | #include "ara/crypto/cryp/decryptor_private_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_29011]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::DigestService |
| Derived from: | std::unique_ptr<DigestService> |
| Syntax: | `using ara::crypto::cryp::DigestService::Uptr = std::unique_ptr<Digest Service>;` |
| Header file: | #include "ara/crypto/cryp/digest_service.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_21001]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::EncryptorPublicCtx |
| Derived from: | std::unique_ptr<EncryptorPublicCtx> |
| Syntax: | `using ara::crypto::cryp::EncryptorPublicCtx::Uptr = std::unique_ ptr<EncryptorPublicCtx>;` |
| Header file: | #include "ara/crypto/cryp/encryptor_public_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02202)*

**[SWS_CRYPT_29042]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::ExtensionService |
| Derived from: | std::unique_ptr<ExtensionService> |
| Syntax: | `using ara::crypto::cryp::ExtensionService::Uptr = std::unique_ ptr<ExtensionService>;` |
| Header file: | #include "ara/crypto/cryp/extension_service.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_21101]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::HashFunctionCtx |
| Derived from: | std::unique_ptr<HashFunctionCtx> |
| Syntax: | `using ara::crypto::cryp::HashFunctionCtx::Uptr = std::unique_ptr<Hash FunctionCtx>;` |
| Header file: | #include "ara/crypto/cryp/hash_function_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02205)*

### [SWS_CRYPT_21301]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Uptr |
| *Scope:* | class ara::crypto::cryp::KeyAgreementPrivateCtx |
| *Derived from:* | std::unique_ptr<KeyAgreementPrivateCtx> |
| *Syntax:* | `using ara::crypto::cryp::KeyAgreementPrivateCtx::Uptr = std::unique_ptr<KeyAgreementPrivateCtx>;` |
| *Header file:* | #include "ara/crypto/cryp/key_agreement_private_ctx.h" |
| *Description:* | Unique smart pointer of this interface. |

⌋*(RS_CRYPTO_02104)*

### [SWS_CRYPT_21401]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Uptr |
| *Scope:* | class ara::crypto::cryp::KeyDecapsulatorPrivateCtx |
| *Derived from:* | std::unique_ptr<KeyDecapsulatorPrivateCtx> |
| *Syntax:* | `using ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Uptr = std::unique_ptr<KeyDecapsulatorPrivateCtx>;` |
| *Header file:* | #include "ara/crypto/cryp/key_decapsulator_private_ctx.h" |
| *Description:* | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02104)*

### [SWS_CRYPT_21501]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Uptr |
| *Scope:* | class ara::crypto::cryp::KeyDerivationFunctionCtx |
| *Derived from:* | std::unique_ptr<KeyDerivationFunctionCtx> |
| *Syntax:* | `using ara::crypto::cryp::KeyDerivationFunctionCtx::Uptr = std::unique_ptr<KeyDerivationFunctionCtx>;` |
| *Header file:* | #include "ara/crypto/cryp/key_derivation_function_ctx.h" |
| *Description:* | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02103)*

### [SWS_CRYPT_21801]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Uptr |
| *Scope:* | class ara::crypto::cryp::KeyEncapsulatorPublicCtx |
| *Derived from:* | std::unique_ptr<KeyEncapsulatorPublicCtx> |
| *Syntax:* | `using ara::crypto::cryp::KeyEncapsulatorPublicCtx::Uptr = std::unique_ptr<KeyEncapsulatorPublicCtx>;` |
| *Header file:* | #include "ara/crypto/cryp/key_encapsulator_public_ctx.h" |
| *Description:* | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02209)*

## [SWS_CRYPT_22101]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::MessageAuthnCodeCtx |
| Derived from: | std::unique_ptr<MessageAuthnCodeCtx> |
| Syntax: | `using ara::crypto::cryp::MessageAuthnCodeCtx::Uptr = std::unique_ptr<MessageAuthnCodeCtx>;` |
| Header file: | #include "ara/crypto/cryp/message_authn_code_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋(*RS_CRYPTO_02203*)

## [SWS_CRYPT_22201]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::MsgRecoveryPublicCtx |
| Derived from: | std::unique_ptr<MsgRecoveryPublicCtx> |
| Syntax: | `using ara::crypto::cryp::MsgRecoveryPublicCtx::Uptr = std::unique_ptr<MsgRecoveryPublicCtx>;` |
| Header file: | #include "ara/crypto/cryp/msg_recovery_public_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋(*RS_CRYPTO_02204*)

## [SWS_CRYPT_22501]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::PrivateKey |
| Derived from: | std::unique_ptr<const PrivateKey> |
| Syntax: | `using ara::crypto::cryp::PrivateKey::Uptrc = std::unique_ptr<const PrivateKey>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/private_key.h" |
| Description: | Unique smart pointer of the interface. |

⌋(*RS_CRYPTO_02306*)

## [SWS_CRYPT_22701]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::PublicKey |
| Derived from: | std::unique_ptr<const PublicKey> |
| Syntax: | `using ara::crypto::cryp::PublicKey::Uptrc = std::unique_ptr<const PublicKey>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/public_key.h" |
| Description: | Unique smart pointer of the interface. |

⌋(*RS_CRYPTO_02202*)

**[SWS_CRYPT_22901]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::RandomGeneratorCtx |
| Derived from: | std::unique_ptr<RandomGeneratorCtx> |
| Syntax: | `using ara::crypto::cryp::RandomGeneratorCtx::Uptr = std::unique_ptr<RandomGeneratorCtx>;` |
| Header file: | #include "ara/crypto/cryp/random_generator_ctx.h" |
| Description: | Shared smart pointer of the interface. |

⌋*(RS_CRYPTO_02206)*

**[SWS_CRYPT_24801]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Usage |
| Scope: | class ara::crypto::cryp::RestrictedUseObject |
| Derived from: | AllowedUsageFlags |
| Syntax: | `using ara::crypto::cryp::RestrictedUseObject::Usage = AllowedUsageFlags;` |
| Header file: | #include "ara/crypto/cryp/cryobj/restricted_use_object.h" |
| Description: | Alias to the container type for bit-flags of allowed usages of the object. |

⌋*(RS_CRYPTO_02008)*

**[SWS_CRYPT_23001]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::SecretSeed |
| Derived from: | std::unique_ptr<const SecretSeed> |
| Syntax: | `using ara::crypto::cryp::SecretSeed::Uptrc = std::unique_ptr<const SecretSeed>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" |
| Description: | Unique smart pointer of a constant interface instance. |

⌋*(RS_CRYPTO_02007)*

**[SWS_CRYPT_23002]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::SecretSeed |
| Derived from: | std::unique_ptr<SecretSeed> |
| Syntax: | `using ara::crypto::cryp::SecretSeed::Uptr = std::unique_ptr<SecretSeed>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" |
| Description: | Unique smart pointer of a volatile interface instance. |

⌋*(RS_CRYPTO_02007)*

### [SWS_CRYPT_23201]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::SigEncodePrivateCtx |
| Derived from: | std::unique_ptr<SigEncodePrivateCtx> |
| Syntax: | `using ara::crypto::cryp::SigEncodePrivateCtx::Uptr = std::unique_ptr<SigEncodePrivateCtx>;` |
| Header file: | #include "ara/crypto/cryp/sig_encode_private_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02202)*

### [SWS_CRYPT_29001]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::SignatureService |
| Derived from: | std::unique_ptr<SignatureService> |
| Syntax: | `using ara::crypto::cryp::SignatureService::Uptr = std::unique_ptr<SignatureService>;` |
| Header file: | #include "ara/crypto/cryp/signature_service.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_23301]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::Signature |
| Derived from: | std::unique_ptr<const Signature> |
| Syntax: | `using ara::crypto::cryp::Signature::Uptrc = std::unique_ptr<const Signature>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/signature.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02203, RS_CRYPTO_02205)*

### [SWS_CRYPT_23501]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::SignerPrivateCtx |
| Derived from: | std::unique_ptr<SignerPrivateCtx> |
| Syntax: | `using ara::crypto::cryp::SignerPrivateCtx::Uptr = std::unique_ptr<SignerPrivateCtx>;` |
| Header file: | #include "ara/crypto/cryp/signer_private_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02204)*

### [SWS_CRYPT_23601]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::StreamCipherCtx |
| Derived from: | std::unique_ptr<StreamCipherCtx> |
| Syntax: | `using ara::crypto::cryp::StreamCipherCtx::Uptr = std::unique_ptr<StreamCipherCtx>;` |
| Header file: | #include "ara/crypto/cryp/stream_cipher_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02201)*

### [SWS_CRYPT_23701]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::SymmetricBlockCipherCtx |
| Derived from: | std::unique_ptr<SymmetricBlockCipherCtx> |
| Syntax: | `using ara::crypto::cryp::SymmetricBlockCipherCtx::Uptr = std::unique_ptr<SymmetricBlockCipherCtx>;` |
| Header file: | #include "ara/crypto/cryp/symmetric_block_cipher_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02201)*

### [SWS_CRYPT_23801]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::cryp::SymmetricKey |
| Derived from: | std::unique_ptr<const SymmetricKey> |
| Syntax: | `using ara::crypto::cryp::SymmetricKey::Uptrc = std::unique_ptr<const SymmetricKey>;` |
| Header file: | #include "ara/crypto/cryp/cryobj/symmetric_key.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02201)*

### [SWS_CRYPT_24001]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::cryp::SymmetricKeyWrapperCtx |
| Derived from: | std::unique_ptr<SymmetricKeyWrapperCtx> |
| Syntax: | `using ara::crypto::cryp::SymmetricKeyWrapperCtx::Uptr = std::unique_ptr<SymmetricKeyWrapperCtx>;` |
| Header file: | #include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02201)*

**[SWS_CRYPT_20506]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mCOType |
| Scope: | struct ara::crypto::cryp::CryptoObject::COIdentifier |
| Type: | CryptoObjectType |
| Syntax: | `CryptoObjectType ara::crypto::cryp::CryptoObject::COIdentifier::m`<br>`COType;` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | type of objext |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_20507]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mCouid |
| Scope: | struct ara::crypto::cryp::CryptoObject::COIdentifier |
| Type: | CryptoObjectUid |
| Syntax: | `CryptoObjectUid ara::crypto::cryp::CryptoObject::COIdentifier::mCouid;` |
| Header file: | #include "ara/crypto/cryp/cryobj/crypto_object.h" |
| Description: | object identifier |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_22503]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kObjectType |
| Scope: | class ara::crypto::cryp::PrivateKey |
| Type: | const CryptoObjectType |
| Syntax: | `static const CryptoObjectType ara::crypto::cryp::PrivateKey::kObject`<br>`Type = CryptoObjectType::kPrivateKey;` |
| Header file: | #include "ara/crypto/cryp/cryobj/private_key.h" |
| Description: | Static mapping of this interface to specific value of CryptoObjectType enumeration. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_22702]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kObjectType |
| Scope: | class ara::crypto::cryp::PublicKey |
| Type: | const CryptoObjectType |
| Syntax: | `static const CryptoObjectType ara::crypto::cryp::PublicKey::kObject`<br>`Type = CryptoObjectType::kPublicKey;` |
| Header file: | #include "ara/crypto/cryp/cryobj/public_key.h" |
| Description: | const object type |

⌋*(RS_CRYPTO_02202)*

### [SWS_CRYPT_23003]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kObjectType |
| Scope: | class ara::crypto::cryp::SecretSeed |
| Type: | const CryptoObjectType |
| Syntax: | `static const CryptoObjectType ara::crypto::cryp::SecretSeed::kObject Type = CryptoObjectType::kSecretSeed;` |
| Header file: | #include "ara/crypto/cryp/cryobj/secret_seed.h" |
| Description: | Static mapping of this interface to specific value of CryptoObjectType enumeration. |

⌋*(RS_CRYPTO_02007)*

### [SWS_CRYPT_23302]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kObjectType |
| Scope: | class ara::crypto::cryp::Signature |
| Type: | const CryptoObjectType |
| Syntax: | `static const CryptoObjectType ara::crypto::cryp::Signature::kObject Type = CryptoObjectType::kSignature;` |
| Header file: | #include "ara/crypto/cryp/cryobj/signature.h" |
| Description: | Signature object initialized. |

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02203, RS_CRYPTO_02205)*

### [SWS_CRYPT_23802]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kObjectType |
| Scope: | class ara::crypto::cryp::SymmetricKey |
| Type: | const CryptoObjectType |
| Syntax: | `static const CryptoObjectType ara::crypto::cryp::SymmetricKey::kObject Type = CryptoObjectType::kSymmetricKey;` |
| Header file: | #include "ara/crypto/cryp/cryobj/symmetric_key.h" |
| Description: | const object type |

⌋*(RS_CRYPTO_02201)*

### [SWS_CRYPT_10101]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mGeneratorUid |
| Scope: | struct ara::crypto::CryptoObjectUid |
| Type: | Uuid |
| Syntax: | `Uuid ara::crypto::CryptoObjectUid::mGeneratorUid;` |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" |
| Description: | UUID of a generator that has produced this COUID. This UUID can be associated with HSM, physical host/ECU or VM. |

⌋*(RS_CRYPTO_02006)*

## 8.2 C++ language binding Key Storage Provider

**[SWS_CRYPT_30400]**{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | class |
| *Symbol:* | KeySlot |
| *Scope:* | namespace ara::crypto::keys |
| *Syntax:* | `class ara::crypto::keys::KeySlot {...};` |
| *Header file:* | #include "ara/crypto/keys/keyslot.h" |
| *Description:* | Key slot port-prototype interface. This class enables access to a physicl key-slot. |

⌋*(RS_CRYPTO_02405)*

**[SWS_CRYPT_30100]**{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | class |
| *Symbol:* | KeyStorageProvider |
| *Scope:* | namespace ara::crypto::keys |
| *Syntax:* | `class ara::crypto::keys::KeyStorageProvider {...};` |
| *Header file:* | #include "ara/crypto/keys/key_storage_provider.h" |
| *Description:* | Key Storage Provider interface. Any object is uniquely identified by the combination of its UUID and type. HSMs/TPMs implementing the concept of "non-extractable keys" should use own copies of externally supplied crypto objects. A few software Crypto Providers can share single key slot if they support same format. |

⌋*(RS_CRYPTO_02109, RS_CRYPTO_02305, RS_CRYPTO_02401)*

**[SWS_CRYPT_30200]**{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | class |
| *Symbol:* | UpdatesObserver |
| *Scope:* | namespace ara::crypto::keys |
| *Syntax:* | `class ara::crypto::keys::UpdatesObserver {...};` |
| *Header file:* | #include "ara/crypto/keys/updates_observer.h" |
| *Description:* | Definition of an "updates observer" interface. <br><br> The "updates observer" interface should be implemented by a consumer application, if a software developer would like to get notifications about the slots' content update events. |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_30405]**{DRAFT} ⌈

| | | |
|---|---|---|
| *Kind:* | function | |
| *Symbol:* | Clear() | |
| *Scope:* | class ara::crypto::keys::KeySlot | |
| *Syntax:* | `virtual ara::core::Result<void> ara::crypto::keys::KeySlot::Clear ()` <br> `noexcept=0;` | |
| *Return value:* | ara::core::Result< void > | – |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |

▽

$\triangle$

| Errors: | CryptoErrorDomain::kUnreserved Resource | if the target slot is not opened writeable. |
|---|---|---|
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Clear the content of this key-slot. This method must perform a secure cleanup without the ability to restore the object data! This method may be used for atomic update of a key slot scoped to some transaction. In such case the the slot will be updated only after correspondent call of CommitTransaction(). | |

⌋*(RS_CRYPTO_02009)*

## [SWS_CRYPT_30510]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | KeySlotContentProps() |
| Scope: | struct ara::crypto::keys::KeySlotContentProps |
| Syntax: | `ara::crypto::keys::KeySlotContentProps::KeySlotContentProps ()=default;` |
| Header file: | #include "ara/crypto/keys/key_slot_content_props.h" |
| Description: | set content properties |

⌋*(RS_CRYPTO_02111)*

## [SWS_CRYPT_30401]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~KeySlot() |
| Scope: | class ara::crypto::keys::KeySlot |
| Syntax: | `virtual ara::crypto::keys::KeySlot::~KeySlot () noexcept=default;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/keys/keyslot.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02405)*

## [SWS_CRYPT_30408]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetContentProps() | |
| Scope: | class ara::crypto::keys::KeySlot | |
| Syntax: | `virtual ara::core::Result<KeySlotContentProps> ara::crypto::keys::KeySlot::GetContentProps () const noexcept=0;` | |
| Return value: | ara::core::Result< KeySlotContent Props > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kEmptyContainer | if the slot is empty |
| | CryptoErrorDomain::kAccessViolation | if this method is called by an Actor, which has no any ("Owner" or "User") access rights to the key slot |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Get an actual properties of a content in the key slot. If this method called by a "User" Actor then always: props.exportability == false. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30403]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | MyProvider() |
| Scope: | class ara::crypto::keys::KeySlot |
| Syntax: | `virtual ara::core::Result<cryp::CryptoProvider::Uptr>`<br>`ara::crypto::keys::KeySlot::MyProvider () const noexcept=0;` |
| Return value: | ara::core::Result< cryp::Crypto Provider::Uptr > | a unique_pointer to the CryptoProvider to be used with this KeySlot |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Retrieve an instance of the CryptoProvider that owns this KeySlot. Any key slot always has an associated default Crypto Provider that can serve this key slot. In the simplest case all key slots can be served by a single Crypto Provider installed on the Adaptive Platform. But in a more complicated case a few different Crypto Providers may coexist in the system, for example if ECU has one or a few HSMs and software cryptography implementation too, and each of them has own physical key storage. In such case different dedicated Crypto Providers may serve mentioned HSMs and the software implementation. . | |

⌋*(RS_CRYPTO_02401)*

## [SWS_CRYPT_30407]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetPrototypedProps() |
| Scope: | class ara::crypto::keys::KeySlot |
| Syntax: | `virtual ara::core::Result<KeySlotPrototypeProps>`<br>`ara::crypto::keys::KeySlot::GetPrototypedProps () const noexcept=0;` |
| Return value: | ara::core::Result< KeySlotPrototype Props > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Get the prototyped properties of the key slot. | |

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30404]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IsEmpty() |
| Scope: | class ara::crypto::keys::KeySlot |
| Syntax: | `virtual bool ara::crypto::keys::KeySlot::IsEmpty () const noexcept=0;` |
| Return value: | bool | true if the slot is empty or false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Check the slot for emptiness. | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_30409]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Open(bool subscribeForUpdates=false, bool writeable=false) | |
| Scope: | class ara::crypto::keys::KeySlot | |
| Syntax: | `virtual ara::core::Result<IOInterface::Uptr> ara::crypto::keys::Key`<br>`Slot::Open (bool subscribeForUpdates=false, bool writeable=false)`<br>`const noexcept=0;` | |
| Parameters (in): | subscribeForUpdates | if this flag is true then the UpdatesObserver instance (previously registered by a call of the method RegisterObserver()) will be subscribed for updates of the opened key slot |
| | writeable | indicates whether the key-slot shall be opened read-only (default) or with write access |
| Return value: | ara::core::Result< IOInterface::Uptr > | an unique smart pointer to the IOInterface associated with the slot content |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidUsage Order | if (true == subscribeForUpdates), but there is no registered instance of the UpdatesObserver in the Key Storage Provider context |
| | CryptoErrorDomain::kBusyResource | if the specified slot is busy because writeable == true but (a) the keyslot is already opened writable, and/or (b) the keyslot is in scope of another ongoing transaction |
| | CryptoErrorDomain::kModified Resource | if the specified slot has been modified after the Key Slot has been opened |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Open this key slot and return an IOInterface to its content. If the UpdatesObserver interface was provided to the call of RegisterObserver() then the UpdatesObserver::OnUpdate() method should be called by Key Storage engine (in a dedicated thread) every time when this slot is updated (and become visible for "Users"). Monitoring of the opened key slot will be continued even after destruction of the returned TrustedContainer, because content of the slot may be loaded to volatile memory (as a CryptoObject or to a CryptoContext of a crypto primitive), but the TrustedContainer may be destroyed after this. Therefore if you need to terminate monitoring of the key slot then you should directly call method UnsubscribeObserver(SlotNumber). | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_30301]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | KeySlotPrototypeProps() |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Syntax: | `ara::crypto::keys::KeySlotPrototypeProps::KeySlotPrototypeProps`<br>`()=default;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |

⌋*(RS_CRYPTO_02110)*

### [SWS_CRYPT_30406]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SaveCopy(const IOInterface &container) | |
| Scope: | class ara::crypto::keys::KeySlot | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::keys::KeySlot::SaveCopy (const IOInterface &container) noexcept=0;` | |
| Parameters (in): | container | the source IOInterface |
| Return value: | ara::core::Result< void > | true if successfully saved |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kIncompatible Object | if the source object has property "session" or if the source IOInterface references a KeySlot from a different CryptoProvider |
| | CryptoErrorDomain::kEmptyContainer | if the source IOInterface is empty |
| | CryptoErrorDomain::kContent Restrictions | if the source object doesn't satisfy the slot restrictions (including version control) |
| | CryptoErrorDomain::kUnreserved Resource | if the target slot is not opened writeable. |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Save the content of a provided source IOInterface to this key-slot. The source container may represent a volatile trusted container or another KeySlot This method may be used for atomic update of a key slot scoped to some transaction. In such case the the slot will be updated only after correspondent call of CommitTransaction(). | |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30220]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const KeySlot &other) | |
| Scope: | class ara::crypto::keys::KeySlot | |
| Syntax: | `KeySlot& ara::crypto::keys::KeySlot::operator= (const KeySlot &other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | KeySlot & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Copy-assign another KeySlot to this instance. | |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30221]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(KeySlot &&other) | |
| Scope: | class ara::crypto::keys::KeySlot | |
| Syntax: | `KeySlot& ara::crypto::keys::KeySlot::operator= (KeySlot &&other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | KeySlot & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/keys/keyslot.h" | |
| Description: | Move-assign another KeySlot to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30123]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | BeginTransaction(const TransactionScope &targetSlots) |
| Scope: | class ara::crypto::keys::KeyStorageProvider |
| Syntax: | virtual ara::core::Result<TransactionId> ara::crypto::keys::KeyStorage Provider::BeginTransaction (const TransactionScope &targetSlots) noexcept=0; |
| Parameters (in): | targetSlots | a list of KeySlots that should be updated during this transaction. |
| Return value: | ara::core::Result< TransactionId > | a unique ID assigned to this transaction |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnreserved Resource | if targetSlots list has a slot that has not been configured with the reserveSpareSlot parameter in the manifest |
| | CryptoErrorDomain::kBusyResource | if targetSlots list has key slots that are already involved to another pending transaction or opened in writing mode |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Begin new transaction for key slots update. In order for a keyslot to be part of a transaction scope, the reserveSpareSlot model parameter of the keyslot has to be set to true. A transaction is dedicated for updating related key slots simultaneously (in an atomic, all-or-nothing, way). All key slots that should be updated by the transaction have to be opened and provided to this function. Any changes to the slots in scope are executed by calling commit(). | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30124]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CommitTransaction(TransactionId id) |
| Scope: | class ara::crypto::keys::KeyStorageProvider |
| Syntax: | virtual ara::core::Result<void> ara::crypto::keys::KeyStorage Provider::CommitTransaction (TransactionId id) noexcept=0; |
| Parameters (in): | id | an ID of a transaction that should be commited |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back) |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Commit changes of the transaction to Key Storage. Any changes of key slots made during a transaction are invisible up to the commit execution. The commit command permanently saves all changes made during the transaction in Key Storage. | |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_30110]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~KeyStorageProvider() |
| Scope: | class ara::crypto::keys::KeyStorageProvider |
| Syntax: | `virtual ara::crypto::keys::KeyStorageProvider::~KeyStorageProvider () noexcept=default;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_30131]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetRegisteredObserver() | |
| Scope: | class ara::crypto::keys::KeyStorageProvider | |
| Syntax: | `virtual UpdatesObserver::Uptr ara::crypto::keys::KeyStorage Provider::GetRegisteredObserver () const noexcept=0;` | |
| Return value: | UpdatesObserver::Uptr | unique pointer to the registered Updates Observer interface (copy of an internal unique pointer is returned, i.e. the Key Storage provider continues to keep the ownership) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Get pointer of registered Updates Observer. The method returns nullptr if no observers have been registered yet! | |

⌋*(RS_CRYPTO_02401)*

**[SWS_CRYPT_30115]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | LoadKeySlot(ara::core::InstanceSpecifier &iSpecify) | |
| Scope: | class ara::crypto::keys::KeyStorageProvider | |
| Syntax: | `virtual ara::core::Result<KeySlot::Uptr> ara::crypto::keys::KeyStorage Provider::LoadKeySlot (ara::core::InstanceSpecifier &iSpecify) noexcept=0;` | |
| Parameters (in): | iSpecify | the target key-slot instance specifier |
| Return value: | ara::core::Result< KeySlot::Uptr > | an unique smart pointer to allocated key slot |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnreserved Resource | if the InstanceSpecifier is incorrect (the slot is not allocated) |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Load a key slot. The functions loads the information associated with a KeySlot into a KeySlot object. | |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30130]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | RegisterObserver(UpdatesObserver::Uptr observer=nullptr) |
| Scope: | class ara::crypto::keys::KeyStorageProvider |
| Syntax: | `virtual UpdatesObserver::Uptr ara::crypto::keys::KeyStorage Provider::RegisterObserver (UpdatesObserver::Uptr observer=nullptr) noexcept=0;` |
| Parameters (in): | observer | optional pointer to a client-supplied Updates Observer instance that should be registered inside Key Storage implementation and called every time, when an opened for usage/loading key slot is updated externally (by its "Owner" application) |
| Return value: | UpdatesObserver::Uptr | unique pointer to previously registered Updates Observer interface (the pointer ownership is "moved out" to the caller code) |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Register consumer Updates Observer. Only one instance of the UpdatesObserver may be registered by an application process, therefore this method always unregister previous observer and return its unique pointer. If (nullptr == observer) then the method only unregister the previous observer! The method returns nullptr if no observers have been registered yet! | |

⌋(*RS_CRYPTO_02401*)

### [SWS_CRYPT_30125]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | RollbackTransaction(TransactionId id) | |
| Scope: | class ara::crypto::keys::KeyStorageProvider | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::keys::KeyStorage Provider::RollbackTransaction (TransactionId id) noexcept=0;` | |
| Parameters (in): | id | an ID of a transaction that should be rolled back |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back) |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Rollback all changes executed during the transaction in Key Storage. The rollback command permanently cancels all changes made during the transaction in Key Storage. A rolled back transaction is completely invisible for all applications. | |

⌋(*RS_CRYPTO_02004*)

### [SWS_CRYPT_30126]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | UnsubscribeObserver(KeySlot &slot) |
| Scope: | class ara::crypto::keys::KeyStorageProvider |

▽

△

| Syntax: | virtual ara::core::Result<void> ara::crypto::keys::KeyStorage Provider::UnsubscribeObserver (KeySlot &slot) noexcept=0; | |
|---|---|---|
| Parameters (in): | slot | number of a slot that should be unsubscribed from the updates observing |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the specified slot is not monitored now (i.e. if it was not successfully opened via OpenAsUser() or it was already unsubscribed by this method) |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Unsubscribe the Update Observer from changes monitoring of the specified slot. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30222]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const KeyStorageProvider &other) | |
| Scope: | class ara::crypto::keys::KeyStorageProvider | |
| Syntax: | KeyStorageProvider& ara::crypto::keys::KeyStorageProvider::operator= (const KeyStorageProvider &other)=default; | |
| Parameters (in): | other | the other instance |
| Return value: | KeyStorageProvider & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Copy-assign another KeyStorageProvider to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30223]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(KeyStorageProvider &&other) | |
| Scope: | class ara::crypto::keys::KeyStorageProvider | |
| Syntax: | KeyStorageProvider& ara::crypto::keys::KeyStorageProvider::operator= ( KeyStorageProvider &&other)=default; | |
| Parameters (in): | other | the other instance |
| Return value: | KeyStorageProvider & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" | |
| Description: | Move-assign another KeyStorageProvider to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30350]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator==(const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) |
| Scope: | namespace ara::crypto::keys |

▽

△

| Syntax: | constexpr bool ara::crypto::keys::operator== (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept; | |
|---|---|---|
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if all members' values of lhs is equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" | |
| Description: | Comparison operator "equal" for KeySlotPrototypeProps operands. | |

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30351]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator!=(const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) | |
| Scope: | namespace ara::crypto::keys | |
| Syntax: | constexpr bool ara::crypto::keys::operator!= (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept; | |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if at least one member of lhs has a value not equal to correspondent member of rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" | |
| Description: | Comparison operator "not equal" for KeySlotPrototypeProps operands. | |

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30550]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator==(const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) | |
| Scope: | namespace ara::crypto::keys | |
| Syntax: | constexpr bool ara::crypto::keys::operator== (const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) noexcept; | |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if all members' values of lhs is equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/keys/key_slot_content_props.h" | |
| Description: | Comparison operator "equal" for KeySlotContentProps operands. | |

⌋*(RS_CRYPTO_02111)*

## [SWS_CRYPT_30551]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator!=(const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) |
| Scope: | namespace ara::crypto::keys |
| Syntax: | `constexpr bool ara::crypto::keys::operator!= (const KeySlotContent Props &lhs, const KeySlotContentProps &rhs) noexcept;` |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if at least one member of lhs has a value not equal to correspondent member of rhs, and false otherwise |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/keys/key_slot_content_props.h" |
| Description: | Comparison operator "not equal" for KeySlotContentProps operands. |

⌋*(RS_CRYPTO_02111)*

## [SWS_CRYPT_30210]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~UpdatesObserver() |
| Scope: | class ara::crypto::keys::UpdatesObserver |
| Syntax: | `virtual ara::crypto::keys::UpdatesObserver::~UpdatesObserver () noexcept=default;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/keys/updates_observer.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30211]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | OnUpdate(const TransactionScope &updatedSlots) |
| Scope: | class ara::crypto::keys::UpdatesObserver |
| Syntax: | `virtual void ara::crypto::keys::UpdatesObserver::OnUpdate (const TransactionScope &updatedSlots) noexcept=0;` |
| Parameters (in): | updatedSlots | List of monitored slots that were updated after opening (for reading) |
| Return value: | None |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/keys/updates_observer.h" |
| Description: | Notification method that should be called if content of specified slots was changed. Key Storage engine should call this method in a dedicated thread. The provided list may include only slots subscribed for observing (during openning with the "User" permissions, i.e. for "reading" via a call of the method OpenAsUser()). Each slot number may present in the provided list only one time! |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30224]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | operator=(const UpdatesObserver &other) |
| *Scope:* | class ara::crypto::keys::UpdatesObserver |
| *Syntax:* | `UpdatesObserver& ara::crypto::keys::UpdatesObserver::operator= (const UpdatesObserver &other)=default;` |
| *Parameters (in):* | other | the other instance |
| *Return value:* | UpdatesObserver & | *this, containing the contents of other |
| *Header file:* | #include "ara/crypto/keys/updates_observer.h" |
| *Description:* | Copy-assign another UpdatesObserver to this instance. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30225]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | operator=(UpdatesObserver &&other) |
| *Scope:* | class ara::crypto::keys::UpdatesObserver |
| *Syntax:* | `UpdatesObserver& ara::crypto::keys::UpdatesObserver::operator= ( UpdatesObserver &&other)=default;` |
| *Parameters (in):* | other | the other instance |
| *Return value:* | UpdatesObserver & | *this, containing the contents of other |
| *Header file:* | #include "ara/crypto/keys/updates_observer.h" |
| *Description:* | Move-assign another UpdatesObserver to this instance. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30500]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | struct |
| *Symbol:* | KeySlotContentProps |
| *Scope:* | namespace ara::crypto::keys |
| *Syntax:* | `struct ara::crypto::keys::KeySlotContentProps {...};` |
| *Header file:* | #include "ara/crypto/keys/key_slot_content_props.h" |
| *Description:* | Properties of current Key Slot Content, i.e. of a current instance stored to the Key Slot. A value of the mAllowedUsage field is bitwise AND of the common usage flags defined at run-time and the usage flags defined by the UserPermissions prototype for current "Actor". |

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02111)*

### [SWS_CRYPT_30511]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Uptr |
| *Scope:* | struct ara::crypto::keys::KeySlotContentProps |
| *Derived from:* | std::unique_ptr<KeySlotContentProps> |
| *Syntax:* | `using ara::crypto::keys::KeySlotContentProps::Uptr = std::unique_ptr<KeySlotContentProps>;` |
| *Header file:* | #include "ara/crypto/keys/key_slot_content_props.h" |
| *Description:* | shared pointer of interface |

⌋*(RS_CRYPTO_02111)*

**[SWS_CRYPT_30300]**{DRAFT} ⌈

| Kind: | struct |
|---|---|
| Symbol: | KeySlotPrototypeProps |
| Scope: | namespace ara::crypto::keys |
| Syntax: | `struct ara::crypto::keys::KeySlotPrototypeProps {...};` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Prototyped Properties of a Key Slot. |

⌋*(RS_CRYPTO_02009, RS_CRYPTO_02110, RS_CRYPTO_02116)*

**[SWS_CRYPT_30302]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Derived from: | std::unique_ptr<KeySlotPrototypeProps> |
| Syntax: | `using ara::crypto::keys::KeySlotPrototypeProps::Uptr = std::unique_ptr<KeySlotPrototypeProps>;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |

⌋*(RS_CRYPTO_02110)*

**[SWS_CRYPT_30402]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::keys::KeySlot |
| Derived from: | std::unique_ptr<KeySlot> |
| Syntax: | `using ara::crypto::keys::KeySlot::Uptr = std::unique_ptr<KeySlot>;` |
| Header file: | #include "ara/crypto/keys/keyslot.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02405)*

**[SWS_CRYPT_30101]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::keys::KeyStorageProvider |
| Derived from: | std::unique_ptr<KeyStorageProvider> |
| Syntax: | `using ara::crypto::keys::KeyStorageProvider::Uptr = std::unique_ptr<KeyStorageProvider>;` |
| Header file: | #include "ara/crypto/keys/key_storage_provider.h" |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30010]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | TransactionId |
| *Scope:* | namespace ara::crypto::keys |
| *Derived from:* | std::uint64_t |
| *Syntax:* | `using ara::crypto::keys::TransactionId = std::uint64_t;` |
| *Header file:* | #include "ara/crypto/keys/elementary_types.h" |
| *Description:* | Definition of a transaction identifier type. The zero value should be reserved for especial cases. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30011]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | TransactionScope |
| *Scope:* | namespace ara::crypto::keys |
| *Derived from:* | ara::core::Vector<KeySlot> |
| *Syntax:* | `using ara::crypto::keys::TransactionScope = ara::core::Vector<Key`<br>`Slot>;` |
| *Header file:* | #include "ara/crypto/keys/elementary_types.h" |
| *Description:* | Definition of a "transaction scope" type. The "transaction scope" defines a list of key slots that are target for update in a transaction. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30201]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Uptr |
| *Scope:* | class ara::crypto::keys::UpdatesObserver |
| *Derived from:* | std::unique_ptr<UpdatesObserver> |
| *Syntax:* | `using ara::crypto::keys::UpdatesObserver::Uptr = std::unique_`<br>`ptr<UpdatesObserver>;` |
| *Header file:* | #include "ara/crypto/keys/updates_observer.h" |
| *Description:* | Shared smart pointer of the interface. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30503]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | mAlgId |
| *Scope:* | struct ara::crypto::keys::KeySlotContentProps |
| *Type:* | CryptoAlgId |
| *Syntax:* | `CryptoAlgId ara::crypto::keys::KeySlotContentProps::mAlgId;` |
| *Header file:* | #include "ara/crypto/keys/key_slot_content_props.h" |
| *Description:* | Cryptoalgorithm of actual object stored to the slot. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_30505]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mObjectSize |
| Scope: | struct ara::crypto::keys::KeySlotContentProps |
| Type: | std::size_t |
| Syntax: | `std::size_t ara::crypto::keys::KeySlotContentProps::mObjectSize;` |
| Header file: | #include "ara/crypto/keys/key_slot_content_props.h" |
| Description: | Actual size of an object currently stored to the slot. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_30508]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mObjectType |
| Scope: | struct ara::crypto::keys::KeySlotContentProps |
| Type: | CryptoObjectType |
| Syntax: | `CryptoObjectType ara::crypto::keys::KeySlotContentProps::mObjectType;` |
| Header file: | #include "ara/crypto/keys/key_slot_content_props.h" |
| Description: | Actual type of an object stored to the slot. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_30501]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mObjectUid |
| Scope: | struct ara::crypto::keys::KeySlotContentProps |
| Type: | CryptoObjectUid |
| Syntax: | `CryptoObjectUid ara::crypto::keys::KeySlotContentProps::mObjectUid;` |
| Header file: | #include "ara/crypto/keys/key_slot_content_props.h" |
| Description: | UID of a Crypto Object stored to the slot. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_30506]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mContentAllowedUsage |
| Scope: | struct ara::crypto::keys::KeySlotContentProps |
| Type: | AllowedUsageFlags |
| Syntax: | `AllowedUsageFlags ara::crypto::keys::KeySlotContentProps::mContent AllowedUsage;` |
| Header file: | #include "ara/crypto/keys/key_slot_content_props.h" |
| Description: | Actual usage restriction flags of an object stored to the slot for the current "Actor". |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_30306]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mAlgId |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | CryptoAlgId |
| Syntax: | `CryptoAlgId ara::crypto::keys::KeySlotPrototypeProps::mAlgId;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Cryptoalgorithm restriction (kAlgIdAny means without restriction). The algorithm can be specified partially: family & length, mode, padding. |

⌋*(RS_CRYPTO_02110)*

### [SWS_CRYPT_30309]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mAllocateSpareSlot |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | bool |
| Syntax: | `bool ara::crypto::keys::KeySlotPrototypeProps::mAllocateSpareSlot;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Indicates whether FC Crypto shall allocate sufficient storage space for a shadow copy of this KeySlot. |

⌋*(RS_CRYPTO_02110)*

### [SWS_CRYPT_30310]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mAllowContentTypeChange |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | bool |
| Syntax: | `bool ara::crypto::keys::KeySlotPrototypeProps::mAllowContentType Change;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Indicates whether the content of this key-slot may be changed, e.g. from storing a symmetric key to storing an RSA key If this is set to false, then the mObjectType of this KeySlotPrototype Props must be a) valid and b) cannot be changed ( i.e. only objects of mObjectType may be stored in this key-slot). |

⌋*(RS_CRYPTO_02110)*

### [SWS_CRYPT_30313]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mContentAllowedUsage |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | AllowedUsageFlags |
| Syntax: | `AllowedUsageFlags ara::crypto::keys::KeySlotPrototypeProps::mContent AllowedUsage;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |

▽

△

| Description: | Indicates how the content may be used. The following use cases of this attribute are considered: |
| --- | --- |
| | the object to be stored in this key-slot has it's AllowedUsageFlags set to kAllowPrototypedOnly. In this case this attribute must be observed when loading the content into a runtime instance (e.g. the AllowedUsageFlags of a SymmetricKey object should be set according to this attribute) mMaxUpdatesAllowed==0, in this case the content is provided during production while the AllowedUsageFlags is modeled using this attribute when this key-slot is flexibly updated the runtime object's AllowedUsageFlags override this attribute upon a later loading from this key-slot |

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30312]{DRAFT} ⌈

| Kind: | variable |
| --- | --- |
| Symbol: | mExportAllowed |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | bool |
| Syntax: | `bool ara::crypto::keys::KeySlotPrototypeProps::mExportAllowed;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Indicates whether the key-slot content may be exported. |

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30311]{DRAFT} ⌈

| Kind: | variable |
| --- | --- |
| Symbol: | mMaxUpdateAllowed |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | std::int32_t |
| Syntax: | `std::int32_t ara::crypto::keys::KeySlotPrototypeProps::mMaxUpdate`<br>`Allowed;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Specifies how many times this key-slot may be updated, e.g.: |
| | a value of 0 means the key-slot content will be pre-set during production a value of 1 means the key-slot content can be updated only once ("OTP") a negative value means the key-slot content can be updated inifinitely |

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30305]{DRAFT} ⌈

| Kind: | variable |
| --- | --- |
| Symbol: | mSlotType |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | KeySlotType |
| Syntax: | `KeySlotType ara::crypto::keys::KeySlotPrototypeProps::mSlotType;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |

▽

△

| Description: | Key-slot type configuration: all key-slots used by the adaptive machine to provide serives such as secure communication, diagnostics, updates, secure storage etc. shall use the type k Machine. All key-slots that will be used by the adaptive user application must use kApplication. A key-manager user application may define kMachine key-slots as well; in this case the integrator must match a corresponding machine key-slot to be managed. |
|---|---|

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30307]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mSlotCapacity |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | std::size_t |
| Syntax: | `std::size_t ara::crypto::keys::KeySlotPrototypeProps::mSlotCapacity;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Capacity of the slot in bytes. |

⌋*(RS_CRYPTO_02110)*

## [SWS_CRYPT_30308]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mObjectType |
| Scope: | struct ara::crypto::keys::KeySlotPrototypeProps |
| Type: | CryptoObjectType |
| Syntax: | `CryptoObjectType ara::crypto::keys::KeySlotPrototypeProps::mObjectType;` |
| Header file: | #include "ara/crypto/keys/key_slot_prototype_props.h" |
| Description: | Restriction of an object type that can be stored the slot. If this field contains CryptoObjectType::kUnknown then without restriction of the type. |

⌋*(RS_CRYPTO_02110)*

## 8.3   C++ language binding X509 Certificate Management Provider

## [SWS_CRYPT_40100]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | BasicCertInfo |
| Scope: | namespace ara::crypto::x509 |
| Base class: | X509Object |
| Syntax: | `class ara::crypto::x509::BasicCertInfo :  public X509Object {...};` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | Basic Certificate Information interface. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40200]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | Certificate |
| Scope: | namespace ara::crypto::x509 |
| Base class: | BasicCertInfo |
| Syntax: | `class ara::crypto::x509::Certificate :  public BasicCertInfo {...};` |
| Header file: | #include "ara/crypto/x509/certificate.h" |
| Description: | X.509 Certificate interface. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40300]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CertSignRequest |
| Scope: | namespace ara::crypto::x509 |
| Base class: | BasicCertInfo |
| Syntax: | `class ara::crypto::x509::CertSignRequest :  public BasicCertInfo {...};` |
| Header file: | #include "ara/crypto/x509/cert_sign_request.h" |
| Description: | Certificate Signing Request (CSR) object interface This interface is dedicated for complete parsing of the request content. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40700]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | OcspRequest |
| Scope: | namespace ara::crypto::x509 |
| Base class: | X509Object |
| Syntax: | `class ara::crypto::x509::OcspRequest :  public X509Object {...};` |
| Header file: | #include "ara/crypto/x509/ocsp_request.h" |
| Description: | On-line Certificate Status Protocol Request. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40800]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | OcspResponse |
| Scope: | namespace ara::crypto::x509 |
| Base class: | X509Object |
| Syntax: | `class ara::crypto::x509::OcspResponse :  public X509Object {...};` |
| Header file: | #include "ara/crypto/x509/ocsp_response.h" |
| Description: | On-line Certificate Status Protocol Response. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_24400]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | X509PublicKeyInfo |
| Scope: | namespace ara::crypto::x509 |
| Base class: | ara::crypto::Serializable |
| Syntax: | `class ara::crypto::x509::X509PublicKeyInfo : public ara::crypto::Serializable {...};` |
| Header file: | #include "ara/crypto/x509/x509_public_key_info.h" |
| Description: | X.509 Public Key Information interface. |

⌋*(RS_CRYPTO_02307)*

### [SWS_CRYPT_40400]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | X509DN |
| Scope: | namespace ara::crypto::x509 |
| Base class: | X509Object |
| Syntax: | `class ara::crypto::x509::X509DN : public X509Object {...};` |
| Header file: | #include "ara/crypto/x509/x509_dn.h" |
| Description: | Interface of X.509 Distinguished Name (DN). |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40500]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | X509Extensions |
| Scope: | namespace ara::crypto::x509 |
| Base class: | X509Object |
| Syntax: | `class ara::crypto::x509::X509Extensions : public X509Object {...};` |
| Header file: | #include "ara/crypto/x509/x509_extensions.h" |
| Description: | Interface of X.509 Extensions. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40900]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | X509Object |
| Scope: | namespace ara::crypto::x509 |
| Base class: | ara::crypto::Serializable |
| Syntax: | `class ara::crypto::x509::X509Object : public ara::crypto::Serializable {...};` |
| Header file: | #include "ara/crypto/x509/x509_object.h" |
| Description: | Common interface of all objects created by X.509 Provider. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40600]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | X509Provider |
| Scope: | namespace ara::crypto::x509 |
| Syntax: | `class ara::crypto::x509::X509Provider {...};` |
| Header file: | #include "ara/crypto/x509/x509_provider.h" |
| Description: | X.509 Provider interface. The X.509 Provider supports two internal storages: volatile (or session) and persistent. All X.509 objects created by the provider should have an actual reference to their parent X.509 Provider. The X.509 Provider can be destroyed only after destroying of all its daughterly objects. Each method of this interface that creates a X.509 object is non-constant, because any such creation increases a references counter of the X.509 Provider. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40932]{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | X509CustomExtensionsParser |
| Scope: | namespace ara::crypto::x509 |
| Syntax: | `class ara::crypto::x509::X509CustomExtensionsParser {...};` |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| Description: | X.509 custom extensions parser Callback class to be implemented by user. Implemented functions get called by X509Provider::ParseCustomCertExtensions when parsing a certificate. If any function of this class returns an error, the parsing will stop. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_24414]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetPublicKey() | |
| Scope: | class ara::crypto::x509::X509PublicKeyInfo | |
| Syntax: | `virtual ara::core::Result<ara::crypto::cryp::PublicKey::Uptrc>` `ara::crypto::x509::X509PublicKeyInfo::GetPublicKey () const` `noexcept=0;` | |
| Return value: | ara::core::Result< ara::crypto::cryp::PublicKey::Uptrc > | unique smart pointer to the created public key of the subject |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/x509_public_key_info.h" | |
| Description: | Get public key object of the subject. Created PublicKey object is session and non-exportable, because generic X.509 certificate or certificate signing request (CSR) doesn't have COUID of the public key, therefore it should be saved or transmitted only as a part of correspondent certificate or CSR. | |

⌋*(RS_CRYPTO_02108, RS_CRYPTO_02306)*

### [SWS_CRYPT_24412]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | GetRequiredHashAlgId() | |
| **Scope:** | class ara::crypto::x509::X509PublicKeyInfo | |
| **Syntax:** | virtual CryptoAlgId ara::crypto::x509::X509PublicKeyInfo::GetRequired HashAlgId () const noexcept=0; | |
| **Return value:** | CryptoAlgId | required hash algorithm ID or kAlgIdAny if the signature algorithm specification does not include a concrete hash function |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/x509/x509_public_key_info.h" | |
| **Description:** | Get an ID of hash algorithm required by current signature algorithm. | |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_24411]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | GetRequiredHashSize() | |
| **Scope:** | class ara::crypto::x509::X509PublicKeyInfo | |
| **Syntax:** | virtual std::size_t ara::crypto::x509::X509PublicKeyInfo::GetRequired HashSize () const noexcept=0; | |
| **Return value:** | std::size_t | required hash size in bytes |
| **Exception Safety:** | noexcept | |
| **Header file:** | #include "ara/crypto/x509/x509_public_key_info.h" | |
| **Description:** | Get the hash size required by current signature algorithm. | |

⌋*(RS_CRYPTO_02309)*

### [SWS_CRYPT_24413]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | GetSignatureSize() | |
| **Scope:** | class ara::crypto::x509::X509PublicKeyInfo | |
| **Syntax:** | virtual std::size_t ara::crypto::x509::X509PublicKeyInfo::GetSignature Size () const noexcept=0; | |
| **Return value:** | std::size_t | size of the signature value in bytes |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/x509/x509_public_key_info.h" | |
| **Description:** | Get size of the signature value produced and required by the current algorithm. | |

⌋*(RS_CRYPTO_02309)*

**[SWS_CRYPT_24410]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetAlgorithmId() |
| Scope: | class ara::crypto::x509::X509PublicKeyInfo |
| Syntax: | `virtual ara::crypto::cryp::CryptoPrimitiveId::Uptrc ara::crypto::x509::X509PublicKeyInfo::GetAlgorithmId ()=0;` |
| Return value: | ara::crypto::cryp::CryptoPrimitive Id::Uptrc | – |
| Header file: | #include "ara/crypto/x509/x509_public_key_info.h" |
| Description: | Get the CryptoPrimitiveId instance of this class. |

⌋*(RS_CRYPTO_02307)*

**[SWS_CRYPT_24415]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IsSameKey(const ara::crypto::cryp::PublicKey &publicKey) |
| Scope: | class ara::crypto::x509::X509PublicKeyInfo |
| Syntax: | `virtual bool ara::crypto::x509::X509PublicKeyInfo::IsSameKey (const ara::crypto::cryp::PublicKey &publicKey) const noexcept=0;` |
| Parameters (in): | publicKey | the public key object for comparison |
| Return value: | bool | true if values of the stored public key and object provided by the argument are identical and false otherwise |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/x509/x509_public_key_info.h" |
| Description: | Verify the sameness of the provided and kept public keys. This method compare the public key values only. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40115]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetConstraints() |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Syntax: | `virtual KeyConstraints ara::crypto::x509::BasicCertInfo::Get Constraints () const noexcept=0;` |
| Return value: | KeyConstraints | key constraints |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | Get the key constraints for the key associated with this PKCS#10 object. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40114]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetPathLimit() |
| **Scope:** | class ara::crypto::x509::BasicCertInfo |
| **Syntax:** | `virtual std::uint32_t ara::crypto::x509::BasicCertInfo::GetPathLimit () const noexcept=0;` |
| **Return value:** | std::uint32_t | certification path length limit |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/basic_cert_info.h" |
| **Description:** | Get the constraint on the path length defined in the Basic Constraints extension. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40113]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | IsCa() |
| **Scope:** | class ara::crypto::x509::BasicCertInfo |
| **Syntax:** | `virtual bool ara::crypto::x509::BasicCertInfo::IsCa () const noexcept=0;` |
| **Return value:** | bool | true if it is a CA request and false otherwise |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/basic_cert_info.h" |
| **Description:** | Check whether the CA attribute of X509v3 Basic Constraints is true (i.e. pathlen=0). |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40112]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | SubjectDn() |
| **Scope:** | class ara::crypto::x509::BasicCertInfo |
| **Syntax:** | `virtual const X509DN& ara::crypto::x509::BasicCertInfo::SubjectDn () const noexcept=0;` |
| **Return value:** | const X509DN & | subject DN |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/basic_cert_info.h" |
| **Description:** | Get the subject DN. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40111]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | SubjectPubKey(cryp::CryptoProvider::Uptr cryptoProvider=nullptr) |
| *Scope:* | class ara::crypto::x509::BasicCertInfo |
| *Syntax:* | `virtual const X509PublicKeyInfo& ara::crypto::x509::BasicCert` `Info::SubjectPubKey (cryp::CryptoProvider::Uptr crypto` `Provider=nullptr) const noexcept=0;` |
| *Parameters (in):* | cryptoProvider | unique pointer of a target Crypto Provider, where the public key will be used |
| *Return value:* | const X509PublicKeyInfo & | constant reference of the subject public key interface |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/x509/basic_cert_info.h" | |
| *Description:* | Load the subject public key information object to realm of specified crypto provider. If (crypto Provider == nullptr) then X509PublicKeyInfo object will be loaded in realm of the Stack-default Crypto Provider. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40217]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | AuthorityKeyId() |
| *Scope:* | class ara::crypto::x509::Certificate |
| *Syntax:* | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >` `ara::crypto::x509::Certificate::AuthorityKeyId () const noexcept=0;` |
| *Return value:* | ara::core::Result< ara::core::Vector< ara::core::Byte > > | size of the DER encoded AuthorityKeyIdentifier in bytes |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Errors:* | CryptoErrorDomain::kInsufficient Capacity | if (id.empty() == false), but its size is not enough for storing the output value |
| *Header file:* | #include "ara/crypto/x509/certificate.h" | |
| *Description:* | Get the DER encoded AuthorityKeyIdentifier of this certificate. If (id.empty() == true) then this method only returns required size of the output buffer. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40215]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | EndTime() |
| *Scope:* | class ara::crypto::x509::Certificate |
| *Syntax:* | `virtual time_t ara::crypto::x509::Certificate::EndTime () const` `noexcept=0;` |
| *Return value:* | time_t | "Not After" of the certificate |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/x509/certificate.h" | |
| *Description:* | Get the "Not After" of the certificate. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40220]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | GetFingerprint(ReadWriteMemRegion fingerprint, cryp::HashFunctionCtx &hashCtx) |
| *Scope:* | class ara::crypto::x509::Certificate |
| *Syntax:* | `virtual ara::core::Result<std::size_t>`<br>`ara::crypto::x509::Certificate::GetFingerprint (ReadWriteMemRegion`<br>`fingerprint, cryp::HashFunctionCtx &hashCtx) const noexcept=0;` |
| *Parameters (in):* | hashCtx | an initialized hash function context |
| *Parameters (out):* | fingerprint | output buffer for the fingerprint storage |
| *Return value:* | ara::core::Result< std::size_t > | number of bytes actually saved to the output buffer |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Errors:* | CryptoErrorDomain::kIncompleteArg State | if the hashCtx context is not initialized |
| *Header file:* | #include "ara/crypto/x509/certificate.h" | |
| *Description:* | Calculate a fingerprint from the whole certificate. The produced fingerprint value saved to the output buffer starting from leading bytes of the hash value. If the capacity of the output buffer is less than the digest size then the digest will be truncated and only leading bytes will be saved. If the capacity of the output buffer is higher than the digest size then only leading bytes of the buffer will be updated. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40221]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | GetStatus() |
| *Scope:* | class ara::crypto::x509::Certificate |
| *Syntax:* | `virtual Status ara::crypto::x509::Certificate::GetStatus () const`<br>`noexcept=0;` |
| *Return value:* | Status | the certificate verification status |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/x509/certificate.h" | |
| *Description:* | Return last verification status of the certificate. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40212]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | IsRoot() |
| *Scope:* | class ara::crypto::x509::Certificate |
| *Syntax:* | `virtual bool ara::crypto::x509::Certificate::IsRoot () const`<br>`noexcept=0;` |
| *Return value:* | bool | true if the TrustMaster has set this certificate as root |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/x509/certificate.h" | |
| *Description:* | Check whether this certificate belongs to a root CA. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40213]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | IssuerDn() |
| Scope: | class ara::crypto::x509::Certificate |
| Syntax: | `virtual const X509DN& ara::crypto::x509::Certificate::IssuerDn () const =0;` |
| Return value: | const X509DN & | Issuer DN of this certificate |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/x509/certificate.h" |
| Description: | Get the issuer certificate DN. |

⌋(*RS_CRYPTO_02306*)

### [SWS_CRYPT_40216]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SerialNumber() |
| Scope: | class ara::crypto::x509::Certificate |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::x509::Certificate::SerialNumber () const noexcept=0;` |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | size of the certificate serial number in bytes |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if (sn.empty() == false), but its size is not enough for storing the output value |
| Header file: | #include "ara/crypto/x509/certificate.h" |
| Description: | Get the serial number of this certificate. If (sn.empty() == true) then this method only returns required size of the output buffer. |

⌋(*RS_CRYPTO_02306*)

### [SWS_CRYPT_40214]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | StartTime() |
| Scope: | class ara::crypto::x509::Certificate |
| Syntax: | `virtual time_t ara::crypto::x509::Certificate::StartTime () const noexcept=0;` |
| Return value: | time_t | "Not Before" of the certificate |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/x509/certificate.h" |
| Description: | Get the "Not Before" of the certificate. |

⌋(*RS_CRYPTO_02306*)

**[SWS_CRYPT_40218]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SubjectKeyId() | |
| Scope: | class ara::crypto::x509::Certificate | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> >`<br>`ara::crypto::x509::Certificate::SubjectKeyId () const noexcept=0;` | |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | size of the DER encoded SubjectKeyIdentifier in bytes |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if (id.empty() == false), but its size is not enough for storing the output value |
| Header file: | #include "ara/crypto/x509/certificate.h" | |
| Description: | Get the DER encoded SubjectKeyIdentifier of this certificate. If (id.empty() == true) then this method only returns required size of the output buffer. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40219]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | VerifyMe(ara::core::Optional< const Certificate::Uptr > caCert) | |
| Scope: | class ara::crypto::x509::Certificate | |
| Syntax: | `virtual bool ara::crypto::x509::Certificate::VerifyMe (`<br>`ara::core::Optional< const Certificate::Uptr > caCert) const`<br>`noexcept=0;` | |
| Parameters (in): | caCert | the optional pointer to a Certification Authority certificate used for signature of the current one |
| Return value: | bool | true if this certificate was verified successfully and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/certificate.h" | |
| Description: | Verify signature of the certificate. Call with (caCert == nullptr) is applicable only if this is a certificate of a root CA. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40211]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | X509Version() | |
| Scope: | class ara::crypto::x509::Certificate | |
| Syntax: | `virtual std::uint32_t ara::crypto::x509::Certificate::X509Version ()`<br>`const noexcept=0;` | |
| Return value: | std::uint32_t | X.509 version |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/certificate.h" | |
| Description: | Get the X.509 version of this certificate object. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40311]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | Verify() |
| **Scope:** | class ara::crypto::x509::CertSignRequest |
| **Syntax:** | `virtual bool ara::crypto::x509::CertSignRequest::Verify () const noexcept=0;` |
| **Return value:** | bool | true if the signature is correct |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/cert_sign_request.h" |
| **Description:** | Verifies self-signed signature of the certificate request. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40313]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | ExportASN1CertSignRequest() |
| **Scope:** | class ara::crypto::x509::CertSignRequest |
| **Syntax:** | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::x509::CertSignRequest::ExportASN1CertSignRequest () noexcept=0;` |
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::Byte > > | a buffer with the formatted CSR |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Errors:** | CryptoErrorDomain::kInvalidUsage Order | this error will be returned in case not all required information has been provided |
| **Header file:** | #include "ara/crypto/x509/cert_sign_request.h" |
| **Description:** | Export this certificate signing request in DER encoded ASN1 format. Note: this is the CSR that can be sent to the CA for obtaining the certificate. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40315]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | GetSignature() |
| **Scope:** | class ara::crypto::x509::CertSignRequest |
| **Syntax:** | `virtual const ara::crypto::cryp::Signature& ara::crypto::x509::Cert SignRequest::GetSignature () const noexcept=0;` |
| **Return value:** | const ara::crypto::cryp::Signature & | signature object of the request |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/cert_sign_request.h" |
| **Description:** | Return signature object of the request. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40314]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | Version() |
| **Scope:** | class ara::crypto::x509::CertSignRequest |
| **Syntax:** | `virtual unsigned ara::crypto::x509::CertSignRequest::Version () const noexcept=0;` |
| **Return value:** | unsigned | format version of the certificate request |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/cert_sign_request.h" |
| **Description:** | Return format version of the certificate request. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40711]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | Version() |
| **Scope:** | class ara::crypto::x509::OcspRequest |
| **Syntax:** | `virtual std::uint32_t ara::crypto::x509::OcspRequest::Version () const noexcept=0;` |
| **Return value:** | std::uint32_t | OCSP request format version |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/ocsp_request.h" |
| **Description:** | Get version of the OCSP request format. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40811]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | Version() |
| **Scope:** | class ara::crypto::x509::OcspResponse |
| **Syntax:** | `virtual std::uint32_t ara::crypto::x509::OcspResponse::Version () const noexcept=0;` |
| **Return value:** | std::uint32_t | OCSP response format version |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/x509/ocsp_response.h" |
| **Description:** | Get version of the OCSP response format. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40413]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetAttribute(AttributeId id) | |
| Scope: | class ara::crypto::x509::X509DN | |
| Syntax: | `virtual ara::core::Result<ara::core::String>`<br>`ara::crypto::x509::X509DN::GetAttribute (AttributeId id) const`<br>`noexcept=0;` | |
| Parameters (in): | id | the identifier of required attribute |
| Return value: | ara::core::Result< ara::core::String > | String of the attribute |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if the id argument has unsupported value |
| | CryptoErrorDomain::kInsufficient Capacity | if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output |
| Header file: | #include "ara/crypto/x509/x509_dn.h" | |
| Description: | Get DN attribute by its ID (this method is applicale to all attributes except kOrgUnit and k DomainComponent). Capacity of the output string must be enough for storing the output value! If (attribute == nullptr) then method only returns required buffer capacity. | |

⌋(*RS_CRYPTO_02306*)

**[SWS_CRYPT_40415]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetAttribute(AttributeId id, unsigned index) | |
| Scope: | class ara::crypto::x509::X509DN | |
| Syntax: | `virtual ara::core::Result<ara::core::String>`<br>`ara::crypto::x509::X509DN::GetAttribute (AttributeId id, unsigned`<br>`index) const noexcept=0;` | |
| Parameters (in): | id | the identifier of required attribute |
| | index | the zero-based index of required component of the attribute |
| Return value: | ara::core::Result< ara::core::String > | String of the attribute |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if the id argument has unsupported value |
| | CryptoErrorDomain::kInsufficient Capacity | if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output |
| | CryptoErrorDomain::kInvalidArgument | if (id != kOrgUnit) && (id != kDomainComponent) && (index > 0) |
| | CryptoErrorDomain::kAboveBoundary | if ((id == kOrgUnit) || (id == kDomainComponent)) and the index value is greater than or equal to the actual number of components in the specified attribute |
| Header file: | #include "ara/crypto/x509/x509_dn.h" | |
| Description: | Return DN attribute by its ID and sequential index (this method is applicale to attributes kOrg Unit and kDomainComponent). Capacity of the output string must be enough for storing the output value! If (attribute == nullptr) then method only returns required buffer capacity. | |

⌋(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40411]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | GetDnString() |
| *Scope:* | class ara::crypto::x509::X509DN |
| *Syntax:* | `virtual ara::core::Result<ara::core::String>`<br>`ara::crypto::x509::X509DN::GetDnString () const noexcept=0;` |
| *Return value:* | ara::core::Result< ara::core::String > | String of the whole DN string |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Errors:* | CryptoErrorDomain::kInsufficient Capacity | if (dn != nullptr), but dn->capacity() is less than required for the output value storing |
| *Header file:* | #include "ara/crypto/x509/x509_dn.h" | |
| *Description:* | Get the whole Distinguished Name (DN) as a single string. Capacity of the output string must be enough for storing the output value! If (dn == nullptr) then method only returns required buffer capacity. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40417]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | operator==(const X509DN &other) |
| *Scope:* | class ara::crypto::x509::X509DN |
| *Syntax:* | `virtual bool ara::crypto::x509::X509DN::operator== (const X509DN`<br>`&other) const noexcept=0;` |
| *Parameters (in):* | other | another instance of DN for comparison |
| *Return value:* | bool | true if the provided DN is identical to this one and false otherwise |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/x509/x509_dn.h" | |
| *Description:* | Check for equality of this and another Distinguished Name (DN) objects. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40418]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | function |
| *Symbol:* | operator!=(const X509DN &other) |
| *Scope:* | class ara::crypto::x509::X509DN |
| *Syntax:* | `bool ara::crypto::x509::X509DN::operator!= (const X509DN &other) const`<br>`noexcept;` |
| *Parameters (in):* | other | another instance of DN for comparison |
| *Return value:* | bool | true if the provided DN is not identical to this one and false otherwise |
| *Exception Safety:* | noexcept | |
| *Thread Safety:* | Thread-safe | |
| *Header file:* | #include "ara/crypto/x509/x509_dn.h" | |
| *Description:* | Check for inequality of this and another Distinguished Name (DN) objects. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40414]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetAttribute(AttributeId id, ara::core::StringView attribute) | |
| Scope: | class ara::crypto::x509::X509DN | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509DN::Set Attribute (AttributeId id, ara::core::StringView attribute) noexcept=0;` | |
| Parameters (in): | id | the identifier of required attributet |
| | attribute | the attribute value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if the id argument has unsupported value |
| | CryptoErrorDomain::kUnexpected Value | if the attribute string contains incorrect characters or it has unsupported length |
| Header file: | #include "ara/crypto/x509/x509_dn.h" | |
| Description: | Set DN attribute by its ID (this method is applicale to all attributes except kOrgUnit and k DomainComponent). | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40416]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SetAttribute(AttributeId id, unsigned index, ara::core::StringView attribute) | |
| Scope: | class ara::crypto::x509::X509DN | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509DN::Set Attribute (AttributeId id, unsigned index, ara::core::StringView attribute) noexcept=0;` | |
| Parameters (in): | id | the identifier of required attribute |
| | index | the zero-based index of required component of the attribute |
| | attribute | the attribute value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnknown Identifier | if the id argument has unsupported value |
| | CryptoErrorDomain::kUnexpected Value | if the attribute string contains incorrect characters or it has unsupported length |
| | CryptoErrorDomain::kInvalidArgument | if (id != kOrgUnit) && (id != kDomainComponent) && (index > 0) |
| | CryptoErrorDomain::kAboveBoundary | if ((id == kOrgUnit) || (id == kDomainComponent)) and the index value is greater than the current number of components in the specified attribute |
| Header file: | #include "ara/crypto/x509/x509_dn.h" | |
| Description: | Set DN attribute by its ID and sequential index (this method is applicale to attributes kOrgUnit and kDomainComponent). | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40412]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetDn(ara::core::StringView dn) |
| Scope: | class ara::crypto::x509::X509DN |
| Syntax: | virtual ara::core::Result<void> ara::crypto::x509::X509DN::SetDn ( ara::core::StringView dn) noexcept=0; |
| Parameters (in): | dn | the single string containing the whole DN value in text format |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/x509_dn.h" | |
| Description: | Set whole Distinguished Name (DN) from a single string. [Error]: CryptoErrorDomain::k UnexpectedValue if the dn string has incorrect syntax. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40511]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | Count() |
| Scope: | class ara::crypto::x509::X509Extensions |
| Syntax: | virtual std::size_t ara::crypto::x509::X509Extensions::Count () const noexcept=0; |
| Return value: | std::size_t | number of elements in the sequence |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/x509_extensions.h" | |
| Description: | Count number of elements in the sequence. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40911]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | MyProvider() |
| Scope: | class ara::crypto::x509::X509Object |
| Syntax: | virtual X509Provider& ara::crypto::x509::X509Object::MyProvider () const noexcept=0; |
| Return value: | X509Provider & | a reference to X.509 Provider instance that provides this object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/x509_object.h" | |
| Description: | Get a reference to X.509 Provider of this object. | |

⌋*(RS_CRYPTO_02401)*

### [SWS_CRYPT_40612]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | BuildDn(ara::core::StringView dn) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual ara::core::Result<X509DN::Uptrc>`<br>`ara::crypto::x509::X509Provider::BuildDn (ara::core::StringView dn)`<br>`noexcept=0;` | |
| Parameters (in): | dn | string representation of the Distinguished Name |
| Return value: | ara::core::Result< X509DN::Uptrc > | unique smart pointer for the created X509DN object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the dn argument has incorrect format |
| | CryptoErrorDomain::kInvalidInputSize | if the dn argument has unsupported length (too large) |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Create completed X.500 Distinguished Name structure from the provided string representation. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40629]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CheckCertStatus(Certificate &cert, const OcspResponse &ocspResponse) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual ara::core::Result<bool> ara::crypto::x509::X509Provider::Check`<br>`CertStatus (Certificate &cert, const OcspResponse &ocspResponse) const`<br>`noexcept=0;` | |
| Parameters (in): | cert | a certificate that should be verified |
| | ocspResponse | an OCSP response |
| Return value: | ara::core::Result< bool > | true if the certificate is verified successfully and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the cert is invalid |
| | CryptoErrorDomain::kRuntimeFault | if the ocspResponse is invalid |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Check certificate status by directly provided OCSP response. This method may be used for implementation of the "OCSP stapling". This method updates the Certificate::Status associated with the certificate. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40630]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CheckCertStatus(const ara::core::Vector< Certificate * > &certList, const OcspResponse &ocspResponse) |
| Scope: | class ara::crypto::x509::X509Provider |

▽

△

| Syntax: | `virtual ara::core::Result<bool> ara::crypto::x509::X509Provider::Check CertStatus (const ara::core::Vector< Certificate * > &certList, const OcspResponse &ocspResponse) const noexcept=0;` | |
|---|---|---|
| Parameters (in): | certList | a certificates list that should be verified |
| | ocspResponse | an OCSP response |
| Return value: | ara::core::Result< bool > | true if the certificates list is verified successfully and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the provided certificates are invalid |
| | CryptoErrorDomain::kRuntimeFault | if the ocspResponse is invalid |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Check status of a certificates list by directly provided OCSP response. This method may be used for implementation of the "OCSP stapling". This method updates the Certificate::Status associated with the certificates in the list. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40639]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CheckCertStatusOnline(Certificate &cert) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual ara::core::Result<Certificate::Status> ara::crypto::x509::X509Provider::CheckCertStatusOnline (Certificate &cert) noexcept=0;` | |
| Parameters (in): | cert | a certificate that should be verified |
| Return value: | ara::core::Result< Certificate::Status > | certificate status This method updates the Certificate::Status associated with the certificate. |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the provided certificate is invalid |
| | CryptoErrorDomain::kNoConnection | if a connection to the OCSP responder cannot be established |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Check certificate status via On-line Certificate Status Protocol (OCSP). | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40635]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CleanupVolatileStorage() |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual void ara::crypto::x509::X509Provider::CleanupVolatileStorage () noexcept=0;` |
| Return value: | None |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/x509/x509_provider.h" |

▽

△

| Description: | Cleanup the volatile certificates storage. After execution of this command the certificates previously imported to the volatile storage cannot be found by a search, but it doesn't influence to already loaded Certificate instances! . |
|---|---|

⌡*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40640]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CreateCertSignRequest(cryp::SignerPrivateCtx::Uptr signerCtx, ReadOnlyMemRegion der SubjectDN, ReadOnlyMemRegion x509Extensions=ReadOnlyMemRegion(), unsigned version=1) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<CertSignRequest::Uptrc>`<br>`ara::crypto::x509::X509Provider::CreateCertSignRequest (cryp::Signer`<br>`PrivateCtx::Uptr signerCtx, ReadOnlyMemRegion derSubjectDN, ReadOnly`<br>`MemRegion x509Extensions=ReadOnlyMemRegion(), unsigned version=1)`<br>`const noexcept=0;` |
| Parameters (in): | signerCtx | the fully-configured SignerPrivateCtx to be used for signing this certificate request |
| | derSubjectDN | the DER-encoded subject distinguished name (DN) of the private key owner |
| | x509Extensions | the DER-encoded X.509 Extensions that should be included to the certification request |
| | version | the format version of the target certification request |
| Return value: | ara::core::Result< CertSign Request::Uptrc > | unique smart pointer to created certification request |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnexpected Value | if any of arguments has incorrect/unsupported value |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Create certification request for a private key loaded to the context. | |

⌡*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40615]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CountCertsInChain(ReadOnlyMemRegion certChain, Serializable::FormatId format Id=Serializable::kFormatDefault) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<std::size_t>`<br>`ara::crypto::x509::X509Provider::CountCertsInChain (ReadOnlyMemRegion`<br>`certChain, Serializable::FormatId formatId=Serializable::kFormat`<br>`Default) const noexcept=0;` |
| Parameters (in): | certChain | DER/PEM-encoded certificate chain (in form of a single BLOB) |
| | formatId | input format identifier (kFormatDefault means auto-detect) |
| Return value: | ara::core::Result< std::size_t > | number of certificates in the chain |
| Exception Safety: | noexcept | |

▽

△

| Thread Safety: | Thread-safe | |
|---|---|---|
| Errors: | CryptoErrorDomain::kInvalidArgument | if the certChain argument cannot be pre-parsed |
| | CryptoErrorDomain::kUnknown Identifier | if the formatId argument has unknown value |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Count number of certificates in a serialized certificate chain represented by a single BLOB. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40611]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateEmptyDn(std::size_t capacity=0) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual ara::core::Result<X509DN::Uptr> ara::crypto::x509::X509Provider::CreateEmptyDn (std::size_t capacity=0) noexcept=0;` | |
| Parameters (in): | capacity | number of bytes that should be reserved for the content of the target X509DN object |
| Return value: | ara::core::Result< X509DN::Uptr > | Unique smart pointer to created empty X509DN object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Create an empty X.500 Distinguished Name (DN) structure. If (0 == capacity) then a maximally supported (by the implementation) capacity must be reserved. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40636]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateEmptyExtensions(std::size_t capacity=0) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual ara::core::Result<X509Extensions::Uptr> ara::crypto::x509::X509Provider::CreateEmptyExtensions (std::size_t capacity=0) noexcept=0;` | |
| Parameters (in): | capacity | number of bytes that should be reserved for the content of the target X509Extensions object |
| Return value: | ara::core::Result< X509Extensions::Uptr > | Shared smart pointer to created empty X509X509Extensions object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Create an empty X.509 Extensions structure. If (0 == capacity) then a maximally supported (by the implementation) capacity must be reserved. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40626]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateOcspRequest(const Certificate &cert, ara::core::Optional< const cryp::SignerPrivateCtx::Uptr > signer) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual ara::core::Result<OcspRequest::Uptrc> ara::crypto::x509::X509Provider::CreateOcspRequest (const Certificate &cert, ara::core::Optional< const cryp::SignerPrivateCtx::Uptr > signer) noexcept=0;` | |
| Parameters (in): | cert | a certificate that should be verified |
| | signer | an optional pointer to initialized signer context (if the request should be signed) |
| Return value: | ara::core::Result< OcspRequest::Uptrc > | unique smart pointer to the created OCSP request |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the provided certificate is invalid |
| | CryptoErrorDomain::kIncompleteArgState | if the signer context is not initialized by a key |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Create OCSP request for specified certificate. This method may be used for implementation of the "OCSP stapling". | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40627]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | CreateOcspRequest(const ara::core::Vector< const Certificate * > &certList, ara::core::Optional< const cryp::SignerPrivateCtx::Uptr > signer) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual ara::core::Result<OcspRequest::Uptrc> ara::crypto::x509::X509Provider::CreateOcspRequest (const ara::core::Vector< const Certificate * > &certList, ara::core::Optional< const cryp::SignerPrivateCtx::Uptr > signer) noexcept=0;` | |
| Parameters (in): | certList | a certificates' list that should be verified |
| | signer | an optional pointer to initialized signer context (if the request should be signed) |
| Return value: | ara::core::Result< OcspRequest::Uptrc > | unique smart pointer to the created OCSP request |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the provided certificates are invalid |
| | CryptoErrorDomain::kIncompleteArgState | if the signer context is not initialized by a key |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Create OCSP request for specified list of certificates. This method may be used for implementation of the "OCSP stapling". | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40613]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | DecodeDn(ReadOnlyMemRegion dn, Serializable::FormatId formatId=Serializable::kFormatDefault) |
| **Scope:** | class ara::crypto::x509::X509Provider |
| **Syntax:** | `virtual ara::core::Result<X509DN::Uptrc>`<br>`ara::crypto::x509::X509Provider::DecodeDn (ReadOnlyMemRegion dn,`<br>`Serializable::FormatId formatId=Serializable::kFormatDefault)`<br>`noexcept=0;` |
| **Parameters (in):** | dn | DER/PEM-encoded representation of the Distinguished Name |
| | formatId | input format identifier (kFormatDefault means auto-detect) |
| **Return value:** | ara::core::Result< X509DN::Uptrc > | unique smart pointer for the created X509DN object |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kInvalidArgument | if the dn argument cannot be parsed |
| | CryptoErrorDomain::kUnknown Identifier | if the formatId argument has unknown value |
| **Header file:** | #include "ara/crypto/x509/x509_provider.h" | |
| **Description:** | Decode X.500 Distinguished Name structure from the provided serialized format. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40631]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | FindCertByDn(const X509DN &subjectDn, const X509DN &issuerDn, time_t validityTimePoint) |
| **Scope:** | class ara::crypto::x509::X509Provider |
| **Syntax:** | `virtual ara::core::Vector<Certificate::Uptrc>`<br>`ara::crypto::x509::X509Provider::FindCertByDn (const X509DN &subject`<br>`Dn, const X509DN &issuerDn, time_t validityTimePoint) noexcept=0;` |
| **Parameters (in):** | subjectDn | subject DN of the target certificate |
| | issuerDn | issuer DN of the target certificate |
| | validityTimePoint | a time point when the target certificate should be valid |
| **Return value:** | ara::core::Vector< Certificate::Uptrc > | a vector of unique smart pointers to found certificates; the vector is empty, if nothing is found |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/x509/x509_provider.h" | |
| **Description:** | Find a certificate by the subject and issuer Distinguished Names (DN). | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40632]{DRAFT} ⌈

| Kind: | function |
|---|---|
| **Symbol:** | FindCertByKeyIds(ReadOnlyMemRegion subjectKeyId, ara::core::Optional< ReadOnlyMem Region > authorityKeyId) |

▽

△

| | |
|---|---|
| ***Scope:*** | class ara::crypto::x509::X509Provider |
| ***Syntax:*** | virtual ara::core::Vector<Certificate::Uptrc> ara::crypto::x509::X509Provider::FindCertByKeyIds (ReadOnlyMemRegion subjectKeyId, ara::core::Optional< ReadOnlyMemRegion > authorityKeyId) noexcept=0; |
| ***Parameters (in):*** | subjectKeyId | subject key identifier (SKID) |
| | authorityKeyId | optional authority key identifier (AKID) |
| ***Return value:*** | ara::core::Vector< Certificate::Uptrc > | a vector of unique smart pointers to found certificates; the vector is empty, if nothing is found k UnknownIdentifier |
| ***Exception Safety:*** | noexcept |
| ***Thread Safety:*** | Thread-safe |
| ***Header file:*** | #include "ara/crypto/x509/x509_provider.h" |
| ***Description:*** | Find a certificate by its SKID & AKID. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40633]{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | function |
| ***Symbol:*** | FindCertBySn(ReadOnlyMemRegion sn, const X509DN &issuerDn) |
| ***Scope:*** | class ara::crypto::x509::X509Provider |
| ***Syntax:*** | virtual ara::core::Result<Certificate::Uptrc> ara::crypto::x509::X509Provider::FindCertBySn (ReadOnlyMemRegion sn, const X509DN &issuerDn) noexcept=0; |
| ***Parameters (in):*** | sn | serial number of the target certificate |
| | issuerDn | authority's Distinguished Names (DN) |
| ***Return value:*** | ara::core::Result< Certificate::Uptrc > | the specified certificate or an error, if the certificate cannot be found |
| ***Exception Safety:*** | noexcept |
| ***Thread Safety:*** | Thread-safe |
| ***Errors:*** | CryptoErrorDomain::kUnknown Identifier | if the specified certificate could not be found |
| ***Header file:*** | #include "ara/crypto/x509/x509_provider.h" |
| ***Description:*** | Find a certificate by its serial number and issue DN. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40634]{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | function |
| ***Symbol:*** | ParseCertSignRequest(ReadOnlyMemRegion csr, bool withMetaData=true) |
| ***Scope:*** | class ara::crypto::x509::X509Provider |
| ***Syntax:*** | virtual ara::core::Result<CertSignRequest::Uptrc> ara::crypto::x509::X509Provider::ParseCertSignRequest (ReadOnlyMem Region csr, bool withMetaData=true) noexcept=0; |
| ***Parameters (in):*** | csr | the buffer containing a certificate signing request |

▽

△

| | withMetaData | specifies the format of the buffer content: TRUE means the object has been previously serialized by using the Serializable interface; FALSE means the CSR was exported using the CertSign Request::ExportASN1CertSignRequest() interface |
|---|---|---|
| **Return value:** | ara::core::Result< CertSign Request::Uptrc > | unique smart pointer to the certificate signing request |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUnsupported Format | is returned in case the provided buffer does not contain the expected format |
| **Header file:** | #include "ara/crypto/x509/x509_provider.h" | |
| **Description:** | Parse a certificate signing request (CSR) provided by the user. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40620]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | ImportCrl(ReadOnlyMemRegion crl) | |
| **Scope:** | class ara::crypto::x509::X509Provider | |
| **Syntax:** | ```virtual ara::core::Result<bool> ara::crypto::x509::X509Provider::ImportCrl (ReadOnlyMemRegion crl) noexcept=0;``` | |
| **Parameters (in):** | crl | serialized CRL or Delta CRL (in form of a BLOB) |
| **Return value:** | ara::core::Result< bool > | true if the CRL is valid and false if it is already expired |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kUnexpected Value | if the provided BLOB is not a CRL/DeltaCRL |
| | CryptoErrorDomain::kRuntimeFault | if the CRL validation has failed |
| **Header file:** | #include "ara/crypto/x509/x509_provider.h" | |
| **Description:** | Import Certificate Revocation List (CRL) or Delta CRL from a memory BLOB. If the imported CRL lists some certificates kept in the persistent or volatile storages then their status must be automatically updated to Status::kInvalid. If some of these certificates were already openned during this operation, then this status change becomes available immediately (via method call Certificate::GetStatus())! All certificates with the status kInvalid should be automatically removed from correspondent storages (immediately if a certificate not in use now or just after its closing otherwise). | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40621]{DRAFT} ⌈

| **Kind:** | function |
|---|---|
| **Symbol:** | Import(const Certificate &cert, ara::core::Optional< ara::core::InstanceSpecifier > iSpecify) |
| **Scope:** | class ara::crypto::x509::X509Provider |
| **Syntax:** | ```virtual ara::core::Result<void> ara::crypto::x509::X509Provider::Import (const Certificate &cert, ara::core::Optional< ara::core::InstanceSpecifier > iSpecify) noexcept=0;``` |

▽

△

| Parameters (in): | cert | a valid certificate that should be imported |
|---|---|---|
| | iSpecify | optionally a valid InstanceSpecifier can be provided that points to a CertificateSlot for persistent storage of the certificate, otherwise the certificate shall be stored in volatile (session) storage |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the provided certificate is invalid |
| | CryptoErrorDomain::kIncompatible Object | if provided certificate has partial collision with a matched CSR in the storage |
| | CryptoErrorDomain::kContent Duplication | if the provided certificate already exists in the storage |
| | CryptoErrorDomain::kAccessViolation | if the InstanceSpecifier points to a CertificateSlot, which the application may only read |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Import the certificate to volatile or persistent storage. Only imported certificate may be found by a search and applied for automatic verifications! A certificate can be imported to only one of storage: volatile or persistent. Therefore if you import a certificate already kept in the persistent storage to the volatile one then nothing changes. But if you import a certificate already kept in the volatile storage to the persistent one then it is "moved" to the persistent realm. If an application successfully imports a certificate that correspond to a CSR existing in the storage then this CSR should be removed. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40641]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | LoadCertificate(ara::core::InstanceSpecifier &iSpecify) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<Certificate::Uptr> ara::crypto::x509::X509Provider::LoadCertificate (ara::core::Instance Specifier &iSpecify) noexcept=0;` |

| Parameters (in): | iSpecify | the target certificate instance specifier |
|---|---|---|
| Return value: | ara::core::Result< Certificate::Uptr > | an unique smart pointer to the instantiated certificate |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kUnreserved Resource | if the InstanceSpecifier is incorrect (the certificate cannot be found) |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Load a certificate from the persistent certificate storage. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40616]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ParseCertChain(ara::core::Vector< Certificate::Uptr > &outcome, ReadOnlyMemRegion cert Chain, Serializable::FormatId formatId=Serializable::kFormatDefault) |
| Scope: | class ara::crypto::x509::X509Provider |

▽

△

| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Provider::Parse CertChain (ara::core::Vector< Certificate::Uptr > &outcome, ReadOnly MemRegion certChain, Serializable::FormatId formatId=Serializable::k FormatDefault) noexcept=0;` | |
|---|---|---|
| **Parameters (in):** | certChain | DER/PEM-encoded certificate chain (in form of a single BLOB) |
| | formatId | input format identifier (kFormatDefault means auto-detect) |
| **Parameters (out):** | outcome | an output vector for imported certificates |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kInsufficient Capacity | if the capacity of outcome vector is less than actual number of certificates in the chain |
| | CryptoErrorDomain::kInvalidArgument | if the certChain argument cannot be parsed |
| | CryptoErrorDomain::kUnknown Identifier | if the formatId argument has unknown value |
| **Header file:** | #include "ara/crypto/x509/x509_provider.h" | |
| **Description:** | Parse a serialized representation of the certificate chain and create their instances. Off-line validation of the parsed certification chain may be done via call VerifyCertChainByCrl(). After validation the certificates may be saved to the session or persistent storage for following search and usage. If the certificates are not imported then they will be lost after destroy of the returned instances! Only imported certificates may be found by a search and applied for automatic verifications! Certificates in the outcome vector will be placed from the root CA certificate (zero index) to the final end-entity certificate (last used index of the vector). | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40617]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| **Symbol:** | ParseCertChain(ara::core::Vector< Certificate::Uptr > &outcome, const ara::core::Vector< Read OnlyMemRegion > &certChain, Serializable::FormatId formatId=Serializable::kFormatDefault) | |
| **Scope:** | class ara::crypto::x509::X509Provider | |
| **Syntax:** | `virtual ara::core::Result<void> ara::crypto::x509::X509Provider::Parse CertChain (ara::core::Vector< Certificate::Uptr > &outcome, const ara::core::Vector< ReadOnlyMemRegion > &certChain, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0;` | |
| **Parameters (in):** | certChain | DER/PEM-encoded certificates chain (each certificate is presented by a separate BLOB in the input vector) |
| | formatId | input format identifier (kFormatDefault means auto-detect) |
| **Parameters (out):** | outcome | output vector of imported certificates |
| **Return value:** | ara::core::Result< void > | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Errors:** | CryptoErrorDomain::kInsufficient Capacity | if capacity of the outcome vector is less than number of elements in the certChain |
| | CryptoErrorDomain::kInvalidArgument | if an element of certChain argument cannot be parsed |
| | CryptoErrorDomain::kUnknown Identifier | if the formatId argument has unknown value |

▽

△

| Header file: | #include "ara/crypto/x509/x509_provider.h" |
|---|---|
| Description: | Parse a serialized representation of the certificate chain and create their instances. Off-line validation of the parsed certification chain may be done via call VerifyCertChainByCrl(). After validation the certificates may be imported to the session or persistent storage for following search and usage. Capacity of the outcome vector must be equal to the size of the certChain vector. If the certificates are not imported then they will be lost after destroy of the returned instances! Only imported certificates may be found by a search and applied for automatic verifications! Certificates in the outcome vector will be placed from the root CA certificate (zero index) to the final end-entity certificate (last used index of the vector). |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40614]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ParseCert(ReadOnlyMemRegion cert, Serializable::FormatId formatId=Serializable::kFormat Default) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<Certificate::Uptr>`<br>`ara::crypto::x509::X509Provider::ParseCert (ReadOnlyMemRegion cert,`<br>`Serializable::FormatId formatId=Serializable::kFormatDefault)`<br>`noexcept=0;` |
| Parameters (in): | cert | DER/PEM-encoded certificate |
| | formatId | input format identifier (kFormatDefault means auto-detect) |
| Return value: | ara::core::Result< Certificate::Uptr > | unique smart pointer to created certificate |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the cert argument cannot be parsed |
| | CryptoErrorDomain::kUnknown Identifier | if the formatId argument has unknown value |
| Header file: | #include "ara/crypto/x509/x509_provider.h" |
| Description: | Parse a serialized representation of the certificate and create its instance. Off-line validation of the parsed certificate may be done via call VerifyCertByCrl(). After validation the certificate may be imported to the session or persistent storage for following search and usage. If the parsed certificate is not imported then it will be lost after destroy of the returned instance! Only imported certificate may be found by a search and applied for automatic verifications! |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40628]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ParseOcspResponse(ReadOnlyMemRegion response) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual OcspResponse::Uptrc ara::crypto::x509::X509Provider::ParseOcsp`<br>`Response (ReadOnlyMemRegion response) const noexcept=0;` |
| Parameters (in): | response | a serialized OCSP response |
| Return value: | OcspResponse::Uptrc | unique smart pointer to the created OCSP response instance |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |

▽

### [SWS_CRYPT_40625]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetAsRootOfTrust(const Certificate &caCert) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Provider::SetAs RootOfTrust (const Certificate &caCert) noexcept=0;` |
| Parameters (in): | caCert | a valid CA certificate that should be trusted |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the provided certificate is invalid |
| | CryptoErrorDomain::kIncompatible Object | if provided certificate doesn't belong to a CA |
| | CryptoErrorDomain::kAccessViolation | if the method called by an application without the "Trust Master" permission |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Set specified CA certificate as a "root of trust". Only a certificate saved to the volatile or persistent storage may be marked as the "root of trust"! Only CA certificate can be a "root of trust"! Multiple certificates on an ECU may be marked as the "root of trust". Only an application with permissions "Trust Master" has the right to call this method! | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40624]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | SetPendingStatus(const CertSignRequest &request) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Provider::Set PendingStatus (const CertSignRequest &request) noexcept=0;` |
| Parameters (in): | request | certificate signing request that should be marked as "pending" |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kAccessViolation | if the method called by an application without the "CA Connector" permission |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Set the "pending" status associated to the CSR that means that the CSR already sent to CA. This method do nothing if the CSR already marked as "pending". Only an application with permissions "CA Connector" has the right to call this method! | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40637]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | UpdateCrlOnline(const Certificate &caCert) |
| Scope: | class ara::crypto::x509::X509Provider |

▽

△

| Syntax: | `virtual ara::core::Result<bool>` `ara::crypto::x509::X509Provider::UpdateCrlOnline (const Certificate &caCert) noexcept=0;` | |
|---|---|---|
| Parameters (in): | caCert | valid CA certificate |
| Return value: | ara::core::Result< bool > | true if the CRL was updated succesfully and false if there is no fresh updates |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInvalidArgument | if the provided certificate is invalid |
| | CryptoErrorDomain::kNoConnection | if the connection cannot be established |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Get Certificate Revocation List (CRL) or Delta CRL via on-line connection. CRL Distribution Point (CRL DP) should be extracted from the provided CA certificate. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40618]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | VerifyCert(Certificate &cert, Certificate::Uptr myRoot=nullptr) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual Certificate::Status ara::crypto::x509::X509Provider::Verify Cert (Certificate &cert, Certificate::Uptr myRoot=nullptr) noexcept=0;` | |
| Parameters (in): | cert | target certificate for verification |
| | myRoot | root certificate to be used for verification - if this is nullptr, use machine root certificates |
| Return value: | Certificate::Status | verification status of the provided certificate |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Verify status of the provided certificate by locally stored CA certificates and CRLs only. This method updates the Certificate::Status associated with the certificate. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40619]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | VerifyCertChain(ara::core::Span< const Certificate::Uptr > chain, Certificate::Uptr my Root=nullptr) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `virtual Certificate::Status ara::crypto::x509::X509Provider::Verify CertChain (ara::core::Span< const Certificate::Uptr > chain, Certificate::Uptr myRoot=nullptr) const noexcept=0;` | |
| Parameters (in): | chain | target certificate chain for verification |
| | myRoot | root certificate to be used for verification - if this is nullptr, use machine root certificates |
| Return value: | Certificate::Status | verification status of the provided certificate chain |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |

▽

△

| Header file: | #include "ara/crypto/x509/x509_provider.h" |
|---|---|
| Description: | Verify status of the provided certification chain by locally stored CA certificates and CRLs only. Verification status of the certificate chain is Certificate::Status::kValid only if all certificates in the chain have such status! Certificates in the chain (presented by container vector) must be placed from the root CA certificate (zero index) to the target end-entity certificate (last used index of the vector). Verification is executed in same order. If the chain verification is failed then status of the first failed certificate is returned. This method updates the Certificate::Status associated with the certificates in the chain. |

⌊*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40914]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ParseCustomCertExtensions(const Certificate &cert, std::unique_ptr< X509CustomExtensions Parser > customExtensionsParser) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Provider::Parse CustomCertExtensions (const Certificate &cert, std::unique_ptr< X509CustomExtensionsParser > customExtensionsParser) const noexcept=0;` |
| Parameters (in): | cert | Certificate object to be parsed |
| | customExtensionsParser | Custom extensions parser that implements the callbacks |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kRuntimeFault | If parsing the extensions fails or calling one of the callback returns an error. |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Parse the custom X.509 extensions This method parses the extensions of the provided certificate and calls the corresponding callbacks of the provided customExtensionsParser for each parsed ASN.1 element. If any call to one of the callbacks returns an error, the parsing stops and returns kRuntimeFault. Parsing starts at the first extension of the certificate and parses all extensions of the certificate. | |

⌊*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40915]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ParseCustomCertExtensions(const Certificate &cert, std::unique_ptr< X509CustomExtensions Parser > customExtensionsParser, X509CustomExtensionsParser::Oid oid) |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Provider::Parse CustomCertExtensions (const Certificate &cert, std::unique_ptr< X509CustomExtensionsParser > customExtensionsParser, X509Custom ExtensionsParser::Oid oid) const noexcept=0;` |
| Parameters (in): | cert | Certificate object to be parsed |
| | customExtensionsParser | Custom extensions parser that implements the callbacks |
| | oid | extension object identifier |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |

▽

△

| Thread Safety: | Thread-safe | |
|---|---|---|
| Errors: | CryptoErrorDomain::kRuntimeFault | If parsing the extensions fails or calling one of the callback returns an error. |
| | CryptoErrorDomain::kUnexpected Value | If the certificate doesn't contain an extension with the provided Oid. |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Parse the custom X.509 extensions This method parses the extension identified by the provided oid of the provided certificate and calls the corresponding callbacks of the provided customExtensionsParser for each parsed ASN.1 element. If any call to one of the callbacks returns an error, the parsing stops and returns kRuntimeFault. Only the sequence of the extension identified by the oid is parsed. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40604]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~X509Provider() |
| Scope: | class ara::crypto::x509::X509Provider |
| Syntax: | `virtual ara::crypto::x509::X509Provider::~X509Provider () noexcept=default;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/x509/x509_provider.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_30226]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const X509Provider &other) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `X509Provider& ara::crypto::x509::X509Provider::operator= (const X509Provider &other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | X509Provider & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/x509/x509_provider.h" | |
| Description: | Copy-assign another X509Provider to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30227]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(X509Provider &&other) | |
| Scope: | class ara::crypto::x509::X509Provider | |
| Syntax: | `X509Provider& ara::crypto::x509::X509Provider::operator= (X509Provider &&other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | X509Provider & | *this, containing the contents of other |

▽

△

| Header file: | #include "ara/crypto/x509/x509_provider.h" |
|---|---|
| Description: | Move-assign another X509Provider to this instance. |

⌋(*RS_CRYPTO_02004*)

## [SWS_CRYPT_40922]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnBitString(BitString parsed_bit_string) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnBitString (BitString parsed_bit_string) noexcept=0;` | |
| Parameters (in): | parsed_bit_string | Parsed bit string value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a bit string is encountered. | |

⌋(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40920]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnBool(bool parsed_bool) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnBool (bool parsed_bool) noexcept=0;` | |
| Parameters (in): | parsed_bool | Parsed boolean value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a boolean is encountered. | |

⌋(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40929]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnGeneralizedTime(GeneralizedTime parsed_generalized_time) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnGeneralizedTime (GeneralizedTime parsed_ generalized_time) noexcept=0;` | |
| Parameters (in): | parsed_generalized_time | Parsed generalized time value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |

▽

△

| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
|---|---|---|
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a generalized time is encountered. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40928]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnIa5String(Ia5String parsed_ia5_string) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnIa5String (Ia5String parsed_ia5_string) noexcept=0;` | |
| Parameters (in): | parsed_ia5_string | Parsed IA5 string value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when an IA5 string is encountered. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40921]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnInteger(Integer parsed_integer) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnInteger (Integer parsed_integer) noexcept=0;` | |
| Parameters (in): | parsed_integer | Parsed integer value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when an integer is encountered. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40924]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnNull() | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnNull () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |

▽

△

| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
|---|---|
| Description: | Called when a NULL is encountered. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40923]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnOctetString(OctetString parsed_octet_string) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnOctetString (OctetString parsed_octet_string) noexcept=0;` | |
| Parameters (in): | parsed_octet_string | Parsed octet string value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when an octet string is encountered. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40925]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnOid(Oid parsed_oid) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnOid (Oid parsed_oid) noexcept=0;` | |
| Parameters (in): | parsed_oid | Parsed oid value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when an oid is encountered. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40931]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnParsingEnd() | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnParsingEnd () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when the parsing is completed. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40927]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnPrintableString(PrintableString parsed_printable_string) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnPrintableString (PrintableString parsed_printable_ string) noexcept=0;` | |
| Parameters (in): | parsed_printable_string | Parsed printable string value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a printable string is encountered. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40917]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnSequenceEnd() | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnSequenceEnd () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a sequence ends. | |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40916]**{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnSequenceStart() | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnSequenceStart () noexcept=0;` | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a sequence starts. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40919]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnSetEnd() | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnSetEnd () noexcept=0; | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a set ends. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40918]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnSetStart() | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnSetStart () noexcept=0; | |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a set starts. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40930]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnUtcTime(UtcTime parsed_utc_time) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnUtcTime (UtcTime parsed_utc_time) noexcept=0; | |
| Parameters (in): | parsed_utc_time | Parsed UTC time value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when a UTC time is encountered. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40926]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | OnUtf8String(Utf8String parsed_utf8_string) | |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser | |
| Syntax: | `virtual ara::core::Result<void> ara::crypto::x509::X509Custom ExtensionsParser::OnUtf8String (Utf8String parsed_utf8_string) noexcept=0;` | |
| Parameters (in): | parsed_utf8_string | Parsed UTF8 string value |
| Return value: | ara::core::Result< void > | – |
| Exception Safety: | noexcept | |
| Errors: | CryptoErrorDomain::kRuntimeFault | Indicates an error to the parser to stop parsing |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" | |
| Description: | Called when an UTF8 string is encountered. | |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40981]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~X509CustomExtensionsParser() |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser |
| Syntax: | `virtual ara::crypto::x509::X509CustomExtensionsParser::~X509Custom ExtensionsParser () noexcept=default;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40101]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | KeyConstraints |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Derived from: | std::uint32_t |
| Syntax: | `using ara::crypto::x509::BasicCertInfo::KeyConstraints = std::uint32_ t;` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | X.509 v3 Key Constraints type definition. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40203]{DRAFT} ⌈

| Kind: | enumeration |
|---|---|
| Symbol: | Status |
| Scope: | class ara::crypto::x509::Certificate |
| Underlying type: | std::uint32_t |
| Syntax: | `enum class Status :  std::uint32_t {...};` |

▽

△

| Values: | kValid= 0 | The certificate is valid. |
|---|---|---|
| | kInvalid= 1 | The certificate is invalid. |
| | kUnknown= 2 | Status of the certificate is unknown yet. |
| | kNoTrust= 3 | The certificate has correct signature, but the ECU has no a root of trust for this certificate. |
| | kExpired= 4 | The certificate has correct signature, but it is already expired (its validity period has ended) |
| | kFuture= 5 | The certificate has correct signature, but its validity period is not started yet. |
| Header file: | #include "ara/crypto/x509/certificate.h" | |
| Description: | Certificate verification status. | |

⌋(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40202]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::x509::Certificate |
| Derived from: | std::unique_ptr<const Certificate> |
| Syntax: | `using ara::crypto::x509::Certificate::Uptrc = std::unique_ptr<const Certificate>;` |
| Header file: | #include "ara/crypto/x509/certificate.h" |
| Description: | Unique smart pointer of the interface. |

⌋(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40201]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::x509::Certificate |
| Derived from: | std::unique_ptr<Certificate> |
| Syntax: | `using ara::crypto::x509::Certificate::Uptr = std::unique_ptr<Certificate>;` |
| Header file: | #include "ara/crypto/x509/certificate.h" |
| Description: | Unique smart pointer of the interface. |

⌋(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40301]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::x509::CertSignRequest |
| Derived from: | std::unique_ptr<const CertSignRequest> |
| Syntax: | `using ara::crypto::x509::CertSignRequest::Uptrc = std::unique_ptr<const CertSignRequest>;` |
| Header file: | #include "ara/crypto/x509/cert_sign_request.h" |
| Description: | Unique smart pointer of the constant interface. |

⌊*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40302]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::x509::CertSignRequest |
| Derived from: | std::unique_ptr<CertSignRequest> |
| Syntax: | `using ara::crypto::x509::CertSignRequest::Uptr = std::unique_ptr<Cert SignRequest>;` |
| Header file: | #include "ara/crypto/x509/cert_sign_request.h" |
| Description: | Unique smart pointer of the interface. |

⌊*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40002]{DRAFT} ⌈

| Kind: | enumeration | |
|---|---|---|
| Symbol: | OcspCertStatus | |
| Scope: | namespace ara::crypto::x509 | |
| Underlying type: | std::uint32_t | |
| Syntax: | `enum class OcspCertStatus :  std::uint32_t {...};` | |
| Values: | kGood= 0 | The certificate is not revoked. |
| | kRevoked= 1 | The certificate has been revoked (either permanantly or temporarily (on hold)) |
| | kUnknown= 2 | The responder doesn't know about the certificate being requested. |
| Header file: | #include "ara/crypto/x509/ocsp_response.h" | |
| Description: | On-line Certificate Status Protocol (OCSP) Certificate Status. | |

⌊*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40702]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::x509::OcspRequest |
| Derived from: | std::unique_ptr<const OcspRequest> |
| Syntax: | `using ara::crypto::x509::OcspRequest::Uptrc = std::unique_ptr<const OcspRequest>;` |
| Header file: | #include "ara/crypto/x509/ocsp_request.h" |
| Description: | Shared smart pointer of the interface. |

⌊*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40701]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::x509::OcspRequest |

▽

△

| Derived from: | std::unique_ptr<OcspRequest> |
|---|---|
| Syntax: | `using ara::crypto::x509::OcspRequest::Uptr = std::unique_ptr<Ocsp Request>;` |
| Header file: | #include "ara/crypto/x509/ocsp_request.h" |
| Description: | Shared smart pointer of the interface. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40001]{DRAFT} ⌈

| Kind: | enumeration | |
|---|---|---|
| Symbol: | OcspResponseStatus | |
| Scope: | namespace ara::crypto::x509 | |
| Underlying type: | std::uint32_t | |
| Syntax: | `enum class OcspResponseStatus :  std::uint32_t {...};` | |
| Values: | kSuccessful= 0 | Response has valid confirmations. |
| | kMalformedRequest= 1 | Illegal confirmation request. |
| | kInternalError= 2 | Internal error in issuer. |
| | kTryLater= 3 | Try again later. |
| | kSigRequired= 5 | Must sign the request. |
| | kUnauthorized= 6 | Request unauthorized. |
| Header file: | #include "ara/crypto/x509/ocsp_response.h" | |
| Description: | On-line Certificate Status Protocol (OCSP) Response Status. | |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40802]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::x509::OcspResponse |
| Derived from: | std::unique_ptr<const OcspResponse> |
| Syntax: | `using ara::crypto::x509::OcspResponse::Uptrc = std::unique_ptr<const OcspResponse>;` |
| Header file: | #include "ara/crypto/x509/ocsp_response.h" |
| Description: | Shared smart pointer of the interface. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40801]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::x509::OcspResponse |
| Derived from: | std::unique_ptr<OcspResponse> |
| Syntax: | `using ara::crypto::x509::OcspResponse::Uptr = std::unique_ptr<Ocsp Response>;` |

▽

△

| Header file: | #include "ara/crypto/x509/ocsp_response.h" |
|---|---|
| Description: | Shared smart pointer of the interface. |

⌟(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40403]{DRAFT} ⌈

| Kind: | enumeration | |
|---|---|---|
| Symbol: | AttributeId | |
| Scope: | class ara::crypto::x509::X509DN | |
| Underlying type: | std::uint32_t | |
| Syntax: | `enum class AttributeId :  std::uint32_t {...};` | |
| Values: | kCommonName= 0 | Common Name. |
| | kCountry= 1 | Country. |
| | kState= 2 | State. |
| | kLocality= 3 | Locality. |
| | kOrganization= 4 | Organization. |
| | kOrgUnit= 5 | Organization Unit. |
| | kStreet= 6 | Street. |
| | kPostalCode= 7 | Postal Code. |
| | kTitle= 8 | Title. |
| | kSurname= 9 | Surname. |
| | kGivenName= 10 | Given Name. |
| | kInitials= 11 | Initials. |
| | kPseudonym= 12 | Pseudonym. |
| | kGenerationQualifier= 13 | Generation Qualifier. |
| | kDomainComponent= 14 | Domain Component. |
| | kDnQualifier= 15 | Distinguished Name Qualifier. |
| | kEmail= 16 | E-mail. |
| | kUri= 17 | URI. |
| | kDns= 18 | DNS. |
| | kHostName= 19 | Host Name (UNSTRUCTUREDNAME) |
| | kIpAddress= 20 | IP Address (UNSTRUCTUREDADDRESS) |
| | kSerialNumbers= 21 | Serial Numbers. |
| | kUserId= 22 | User ID. |
| Header file: | #include "ara/crypto/x509/x509_dn.h" | |
| Description: | Enumeration of DN attributes' identifiers. | |

⌟(*RS_CRYPTO_02306*)

## [SWS_CRYPT_40402]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::x509::X509DN |
| Derived from: | std::unique_ptr<const X509DN> |

▽

△

| Syntax: | using ara::crypto::x509::X509DN::Uptrc = std::unique_ptr<const X509DN>; |
|---|---|
| Header file: | #include "ara/crypto/x509/x509_dn.h" |
| Description: | Unique smart pointer of the constant interface. |

⌋(RS_CRYPTO_02306)

## [SWS_CRYPT_40401]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::x509::X509DN |
| Derived from: | std::unique_ptr<X509DN> |
| Syntax: | using ara::crypto::x509::X509DN::Uptr = std::unique_ptr<X509DN>; |
| Header file: | #include "ara/crypto/x509/x509_dn.h" |
| Description: | Unique smart pointer of the interface. |

⌋(RS_CRYPTO_02306)

## [SWS_CRYPT_40501]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::x509::X509Extensions |
| Derived from: | std::unique_ptr<X509Extensions> |
| Syntax: | using ara::crypto::x509::X509Extensions::Uptr = std::unique_ptr<X509Extensions>; |
| Header file: | #include "ara/crypto/x509/x509_extensions.h" |
| Description: | Shared smart pointer of the interface. |

⌋(RS_CRYPTO_02306)

## [SWS_CRYPT_24401]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::x509::X509PublicKeyInfo |
| Derived from: | std::unique_ptr<const X509PublicKeyInfo> |
| Syntax: | using ara::crypto::x509::X509PublicKeyInfo::Uptrc = std::unique_ptr<const X509PublicKeyInfo>; |
| Header file: | #include "ara/crypto/x509/x509_public_key_info.h" |
| Description: | Unique smart pointer of the interface. |

⌋(RS_CRYPTO_02307)

### [SWS_CRYPT_40601]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Uptr |
| *Scope:* | class ara::crypto::x509::X509Provider |
| *Derived from:* | std::unique_ptr<X509Provider> |
| *Syntax:* | `using ara::crypto::x509::X509Provider::Uptr = std::unique_ptr<X509Provider>;` |
| *Header file:* | #include "ara/crypto/x509/x509_provider.h" |
| *Description:* | Shared smart pointer of the interface. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40602]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | StorageIndex |
| *Scope:* | class ara::crypto::x509::X509Provider |
| *Derived from:* | std::size_t |
| *Syntax:* | `using ara::crypto::x509::X509Provider::StorageIndex = std::size_t;` |
| *Header file:* | #include "ara/crypto/x509/x509_provider.h" |
| *Description:* | Type of an internal index inside the certificate storage. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40935]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | BitString |
| *Scope:* | class ara::crypto::x509::X509CustomExtensionsParser |
| *Derived from:* | std::pair<ara::crypto::ReadOnlyMemRegion, NumberOfUnusedBits> |
| *Syntax:* | `using ara::crypto::x509::X509CustomExtensionsParser::BitString = std::pair<ara::crypto::ReadOnlyMemRegion, NumberOfUnusedBits>;` |
| *Header file:* | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| *Description:* | Type alias. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40941]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | GeneralizedTime |
| *Scope:* | class ara::crypto::x509::X509CustomExtensionsParser |
| *Derived from:* | ara::core::StringView |
| *Syntax:* | `using ara::crypto::x509::X509CustomExtensionsParser::GeneralizedTime = ara::core::StringView;` |
| *Header file:* | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| *Description:* | Type alias. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40940]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Ia5String |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser |
| Derived from: | ara::core::StringView |
| Syntax: | `using ara::crypto::x509::X509CustomExtensionsParser::Ia5String = ara::core::StringView;` |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| Description: | Type alias. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40933]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Integer |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser |
| Derived from: | ara::crypto::ReadOnlyMemRegion |
| Syntax: | `using ara::crypto::x509::X509CustomExtensionsParser::Integer = ara::crypto::ReadOnlyMemRegion;` |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| Description: | Type alias. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40934]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | NumberOfUnusedBits |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser |
| Derived from: | std::uint8_t |
| Syntax: | `using ara::crypto::x509::X509CustomExtensionsParser::NumberOfUnusedBits = std::uint8_t;` |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| Description: | Type alias. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40936]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | OctetString |
| Scope: | class ara::crypto::x509::X509CustomExtensionsParser |
| Derived from: | ara::crypto::ReadOnlyMemRegion |
| Syntax: | `using ara::crypto::x509::X509CustomExtensionsParser::OctetString = ara::crypto::ReadOnlyMemRegion;` |
| Header file: | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| Description: | Type alias. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40937]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Oid |
| *Scope:* | class ara::crypto::x509::X509CustomExtensionsParser |
| *Derived from:* | ara::core::StringView |
| *Syntax:* | using ara::crypto::x509::X509CustomExtensionsParser::Oid = ara::core::StringView; |
| *Header file:* | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| *Description:* | Type alias. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40939]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | PrintableString |
| *Scope:* | class ara::crypto::x509::X509CustomExtensionsParser |
| *Derived from:* | ara::core::StringView |
| *Syntax:* | using ara::crypto::x509::X509CustomExtensionsParser::PrintableString = ara::core::StringView; |
| *Header file:* | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| *Description:* | Type alias. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40942]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | UtcTime |
| *Scope:* | class ara::crypto::x509::X509CustomExtensionsParser |
| *Derived from:* | ara::core::StringView |
| *Syntax:* | using ara::crypto::x509::X509CustomExtensionsParser::UtcTime = ara::core::StringView; |
| *Header file:* | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| *Description:* | Type alias. |

⌋*(RS_CRYPTO_02306)*

### [SWS_CRYPT_40938]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | type alias |
| *Symbol:* | Utf8String |
| *Scope:* | class ara::crypto::x509::X509CustomExtensionsParser |
| *Derived from:* | ara::crypto::ReadOnlyMemRegion |
| *Syntax:* | using ara::crypto::x509::X509CustomExtensionsParser::Utf8String = ara::crypto::ReadOnlyMemRegion; |
| *Header file:* | #include "ara/crypto/x509/x509_custom_extension_parser.h" |
| *Description:* | Type alias. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40157]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrCrlSign |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrCrlSign = 0x0200; |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The key can be used for Certificates Revokation Lists (CRL) signing. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40154]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrDataEncipherment |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDataEncipherment = 0x1000; |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The key can be used for data encipherment. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40159]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrDecipherOnly |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDecipherOnly = 0x0080; |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The enciphermet key can be used for deciphering only. |

⌋*(RS_CRYPTO_02306)*

## [SWS_CRYPT_40151]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrDigitalSignature |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDigitalSignature = 0x8000; |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The key can be used for digital signature production. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40158]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrEncipherOnly |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | `static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrEncipherOnly = 0x0100;` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The enciphermet key can be used for enciphering only. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40155]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrKeyAgreement |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | `static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyAgreement = 0x0800;` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The key can be used for a key agreement protocol execution. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40156]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrKeyCertSign |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | `static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyCertSign = 0x0400;` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The key can be used for certificates signing. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40153]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrKeyEncipherment |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | `static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyEncipherment = 0x2000;` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The key can be used for key encipherment. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40152]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrNonRepudiation |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | `static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstr NonRepudiation = 0x4000;` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | The key can be used in cases requiring the "non-repudiation" guarantee. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40150]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kConstrNone |
| Scope: | class ara::crypto::x509::BasicCertInfo |
| Type: | const KeyConstraints |
| Syntax: | `static const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstr None = 0;` |
| Header file: | #include "ara/crypto/x509/basic_cert_info.h" |
| Description: | No key constraints. |

⌋*(RS_CRYPTO_02306)*

**[SWS_CRYPT_40603]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kInvalidIndex |
| Scope: | class ara::crypto::x509::X509Provider |
| Type: | const StorageIndex |
| Syntax: | `static const StorageIndex ara::crypto::x509::X509Provider::kInvalid Index = static_cast<std::size_t>(-1LL);` |
| Header file: | #include "ara/crypto/x509/x509_provider.h" |
| Description: | Reserved "invalid index" value for navigation inside the certificate storage. |

⌋*(RS_CRYPTO_02306)*

## 8.4 API Common Data Types

**[SWS_CRYPT_10015]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | AllowedUsageFlags |
| Scope: | namespace ara::crypto |
| Derived from: | std::uint32_t |

▽

$\triangle$

| Syntax: | using ara::crypto::AllowedUsageFlags = std::uint32_t; |
|---|---|
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | A container type and constant bit-flags of allowed usages of a key or a secret seed object. Only directly specified usages of a key are allowed, all other are prohibited! Similar set of flags are defined for the usage restrictions of original key/seed and for a symmetric key or seed that potentially can be derived from the original one. A symmetric key or secret seed can be derived from the original one, only if it supports kAllowKeyAgreement or kAllowKeyDiversify or kAllow KeyDerivation! |

⌋(*RS_CRYPTO_02111*)

## [SWS_CRYPT_10042]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | ByteVector |
| Scope: | namespace ara::crypto |
| Derived from: | ara::core::Vector<std::uint8_t, Alloc> |
| Syntax: | using ara::crypto::ByteVector = ara::core::Vector<std::uint8_t, Alloc>; |
| Template param: | Alloc | custom allocator of bytes sequences |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | Alias of a bytes' vector template with customizable allocator. |

⌋(*RS_CRYPTO_02201*, *RS_CRYPTO_02202*, *RS_CRYPTO_02203*, *RS_CRYPTO_-02204*, *RS_CRYPTO_02205*, *RS_CRYPTO_02206*, *RS_CRYPTO_02207*, *RS_-CRYPTO_02208*, *RS_CRYPTO_02209*)

## [SWS_CRYPT_10014]{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | CryptoAlgId |
| Scope: | namespace ara::crypto |
| Derived from: | std::uint64_t |
| Syntax: | using ara::crypto::CryptoAlgId = std::uint64_t; |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | Container type of the Crypto Algorithm Identifier. |

⌋(*RS_CRYPTO_02102*, *RS_CRYPTO_02107*)

## [SWS_CRYPT_10016]{DRAFT} ⌈

| Kind: | enumeration | |
|---|---|---|
| Symbol: | CryptoObjectType | |
| Scope: | namespace ara::crypto | |
| Underlying type: | std::uint32_t | |
| Syntax: | enum class CryptoObjectType :  std::uint32_t {...}; | |
| Values: | kUndefined= 0 | Object type is currently not defined (empty container) |
| | kSymmetricKey= 1 | cryp::SymmetricKey object |

$\triangledown$

△

| | kPrivateKey= 2 | cryp::PrivateKey object |
|---|---|---|
| | kPublicKey= 3 | cryp::PublicKey object |
| | kSignature= 4 | cryp::Signature object (asymmetric digital signature or symmetric MAC/HMAC or hash digest) |
| | kSecretSeed= 5 | cryp::SecretSeed object. Note: the seed cannot have an associated crypto algorithm! |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" | |
| *Description:* | Enumeration of all types of crypto objects, i.e. types of content that can be stored to a key slot. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_10100]{DRAFT} ⌈

| *Kind:* | struct |
|---|---|
| *Symbol:* | CryptoObjectUid |
| *Scope:* | namespace ara::crypto |
| *Syntax:* | `struct ara::crypto::CryptoObjectUid {...};` |
| *Header file:* | #include "ara/crypto/common/crypto_object_uid.h" |
| *Description:* | Definition of Crypto Object Unique Identifier (COUID) type. |

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

## [SWS_CRYPT_10017]{DRAFT} ⌈

| *Kind:* | enumeration | |
|---|---|---|
| *Symbol:* | ProviderType | |
| *Scope:* | namespace ara::crypto | |
| *Underlying type:* | std::uint32_t | |
| *Syntax:* | `enum class ProviderType :  std::uint32_t {...};` | |
| *Values:* | kUndefinedProvider= 0 | Undefined/Unknown Provider type (or applicable for the whole Crypto Stack) |
| | kCryptoProvider= 1 | Cryptography Provider. |
| | kKeyStorageProvider= 2 | Key Storage Provider. |
| | kX509Provider= 3 | X.509 Provider. |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" | |
| *Description:* | Enumeration of all known Provider types. | |

⌋*(RS_CRYPTO_02401, RS_CRYPTO_02109)*

## [SWS_CRYPT_10033]{DRAFT} ⌈

| *Kind:* | type alias |
|---|---|
| *Symbol:* | ReadOnlyMemRegion |
| *Scope:* | namespace ara::crypto |
| *Derived from:* | ara::core::Span<const std::uint8_t> |
| *Syntax:* | `using ara::crypto::ReadOnlyMemRegion = ara::core::Span<const std::uint8_t>;` |
| *Header file:* | #include "ara/crypto/common/mem_region.h" |
| *Description:* | Read-Only Memory Region (intended for [in] arguments) |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_10031]**{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | type alias |
| ***Symbol:*** | ReadWriteMemRegion |
| ***Scope:*** | namespace ara::crypto |
| ***Derived from:*** | ara::core::Span<std::uint8_t> |
| ***Syntax:*** | `using ara::crypto::ReadWriteMemRegion = ara::core::Span<std::uint8_t>;` |
| ***Header file:*** | #include "ara/crypto/common/mem_region.h" |
| ***Description:*** | Read-Write Memory Region (intended for [in/out] arguments) |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_10099]**{DRAFT} ⌈

| | | |
|---|---|---|
| ***Kind:*** | enumeration | |
| ***Symbol:*** | CryptoErrc | |
| ***Scope:*** | namespace ara::crypto | |
| ***Underlying type:*** | ara::core::ErrorDomain::CodeType | |
| ***Syntax:*** | `enum class CryptoErrc :  ara::core::ErrorDomain::CodeType {...};` | |
| ***Values:*** | kErrorClass= 0x1000000 | Reserved (a multiplier of error class IDs) |
| | kErrorSubClass= 0x10000 | Reserved (a multiplier of error sub-class IDs) |
| | kErrorSubSubClass= 0x100 | Reserved (a multiplier of error sub-sub-class IDs) |
| | kResourceFault= 1 * kErrorClass | ResourceException: Generic resource fault! |
| | kBusyResource= kResourceFault + 1 | ResourceException: Specified resource is busy! |
| | kInsufficientResource= kResourceFault + 2 | ResourceException: Insufficient capacity of specified resource! |
| | kUnreservedResource= kResourceFault + 3 | ResourceException: Specified resource was not reserved! |
| | kModifiedResource= kResourceFault + 4 | ResourceException: Specified resource has been modified! |
| | kLogicFault= 2 * kErrorClass | LogicException: Generic logic fault! |
| | kInvalidArgument= kLogicFault + 1 * kErrorSubClass | InvalidArgumentException: An invalid argument value is provided! |
| | kUnknownIdentifier= kInvalidArgument + 1 | InvalidArgumentException: Unknown identifier is provided! |
| | kInsufficientCapacity= kInvalidArgument + 2 | InvalidArgumentException: Insufficient capacity of the output buffer! |
| | kInvalidInputSize= kInvalidArgument + 3 | InvalidArgumentException: Invalid size of an input buffer! |
| | kIncompatibleArguments= kInvalidArgument + 4 | InvalidArgumentException: Provided values of arguments are incompatible! |
| | kInOutBuffersIntersect= kInvalidArgument + 5 | InvalidArgumentException: Input and output buffers are intersect! |
| | kBelowBoundary= kInvalidArgument + 6 | InvalidArgumentException: Provided value is below the lower boundary! |
| | kAboveBoundary= kInvalidArgument + 7 | InvalidArgumentException: Provided value is above the upper boundary! |
| | kAuthTagNotValid= kInvalidArgument + 8 | AuthTagNotValidException: Provided authentication-tag cannot be verified! |

▽

△

| | kUnsupported= kInvalidArgument + 1 * kErrorSubSubClass | UnsupportedException: Unsupported request (due to limitations of the implementation)! |
|---|---|---|
| | kInvalidUsageOrder= kLogicFault + 2 * kErrorSubClass | InvalidUsageOrderException: Invalid usage order of the interface! |
| | kUninitializedContext= kInvalidUsage Order + 1 | InvalidUsageOrderException: Context of the interface was not initialized! |
| | kProcessingNotStarted= kInvalidUsage Order + 2 | InvalidUsageOrderException: Data processing was not started yet! |
| | kProcessingNotFinished= kInvalid UsageOrder + 3 | InvalidUsageOrderException: Data processing was not finished yet! |
| | kRuntimeFault= 3 * kErrorClass | RuntimeException: Generic runtime fault! |
| | kUnsupportedFormat= kRuntimeFault + 1 | RuntimeException: Unsupported serialization format for this object type! |
| | kBruteForceRisk= kRuntimeFault + 2 | RuntimeException: Operation is prohibitted due to a risk of a brute force attack! |
| | kContentRestrictions= kRuntimeFault + 3 | RuntimeException: The operation violates content restrictions of the target container! |
| | kBadObjectReference= kRuntimeFault + 4 | RuntimeException: Incorrect reference between objects! |
| | kContentDuplication= kRuntimeFault + 6 | RuntimeException: Provided content already exists in the target storage! |
| | kUnexpectedValue= kRuntimeFault + 1 * kErrorSubClass | UnexpectedValueException: Unexpected value of an argument is provided! |
| | kIncompatibleObject= kUnexpected Value + 1 | UnexpectedValueException: The provided object is incompatible with requested operation or its configuration! |
| | kIncompleteArgState= kUnexpected Value + 2 | UnexpectedValueException: Incomplete state of an argument! |
| | kEmptyContainer= kUnexpectedValue + 3 | UnexpectedValueException: Specified container is empty! |
| | kMissingArgument= kUnexpectedValue + 4 | kMissingArgumentException: Expected argument, but none provided! |
| | kBadObjectType= kUnexpectedValue + 1 * kErrorSubSubClass | BadObjectTypeException: Provided object has unexpected type! |
| | kUsageViolation= kRuntimeFault + 2 * kErrorSubClass | UsageViolationException: Violation of allowed usage for the object! |
| | kAccessViolation= kRuntimeFault + 3 * kErrorSubClass | AccessViolationException: Access rights violation! |
| **Header file:** | #include "ara/crypto/common/crypto_error_domain.h" | |
| **Description:** | Enumeration of all Crypto Error Code values that may be reported by ara::crypto. | |

⌋*(RS_CRYPTO_02310)*

## [SWS_CRYPT_30001]{DRAFT} ⌈

| **Kind:** | struct |
|---|---|
| **Symbol:** | SecureCounter |
| **Scope:** | namespace ara::crypto |
| **Syntax:** | `struct ara::crypto::SecureCounter {...};` |
| **Header file:** | #include "ara/crypto/common/entry_point.h" |
| **Description:** | 128 bit secure counter made up of most significant and least significant quad-word of the hardware counter. |

⌋*(RS_CRYPTO_02401)*

## [SWS_CRYPT_10701]{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | type alias |
| ***Symbol:*** | FormatId |
| ***Scope:*** | class ara::crypto::Serializable |
| ***Derived from:*** | std::uint32_t |
| ***Syntax:*** | `using ara::crypto::Serializable::FormatId = std::uint32_t;` |
| ***Header file:*** | #include "ara/crypto/common/serializable.h" |
| ***Description:*** | A container type for the encoding format identifiers. |

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02302)*

## [SWS_CRYPT_10019]{DRAFT} ⌈

| | | |
|---|---|---|
| ***Kind:*** | enumeration | |
| ***Symbol:*** | CryptoTransform | |
| ***Scope:*** | namespace ara::crypto | |
| ***Underlying type:*** | std::uint32_t | |
| ***Syntax:*** | `enum class CryptoTransform :  std::uint32_t {...};` | |
| ***Values:*** | kEncrypt= 1 | encryption |
| | kDecrypt= 2 | decryption |
| | kMacVerify= 3 | MAC verification. |
| | kMacGenerate= 4 | MAC generation. |
| | kWrap= 5 | key wrapping |
| | kUnwrap= 6 | key unwrapping |
| | kSigVerify= 7 | signature verification |
| | kSigGenerate= 8 | signature generation |
| ***Header file:*** | #include "ara/crypto/common/base_id_types.h" | |
| ***Description:*** | Enumeration of cryptographic transformations. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_10852]{DRAFT} ⌈

| | |
|---|---|
| ***Kind:*** | type alias |
| ***Symbol:*** | Uptr |
| ***Scope:*** | class ara::crypto::VolatileTrustedContainer |
| ***Derived from:*** | std::unique_ptr<VolatileTrustedContainer> |
| ***Syntax:*** | `using ara::crypto::VolatileTrustedContainer::Uptr = std::unique_ptr<VolatileTrustedContainer>;` |
| ***Header file:*** | #include "ara/crypto/common/volatile_trusted_container.h" |
| ***Description:*** | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_10400]**{DRAFT} ⌈

| Kind: | struct |
|---|---|
| Symbol: | Uuid |
| Scope: | namespace ara::crypto |
| Syntax: | `struct ara::crypto::Uuid {...};` |
| Header file: | #include "ara/crypto/common/uuid.h" |
| Description: | Definition of Universally Unique Identifier (UUID) type. Independently from internal definition details of this structure, it's size must be 16 bytes and entropy of this ID should be close to 128 bit! |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_10801]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptr |
| Scope: | class ara::crypto::IOInterface |
| Derived from: | std::unique_ptr<IOInterface> |
| Syntax: | `using ara::crypto::IOInterface::Uptr = std::unique_ptr<IOInterface>;` |
| Header file: | #include "ara/crypto/common/io_interface.h" |
| Description: | Unique smart pointer of the interface. |

⌋*(RS_CRYPTO_02109)*

**[SWS_CRYPT_10802]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Uptrc |
| Scope: | class ara::crypto::IOInterface |
| Derived from: | std::unique_ptr<const IOInterface> |
| Syntax: | `using ara::crypto::IOInterface::Uptrc = std::unique_ptr<const IOInterface>;` |
| Header file: | #include "ara/crypto/common/io_interface.h" |
| Description: | Unique smart pointer of the constant interface. |

⌋*(RS_CRYPTO_02109)*

**[SWS_CRYPT_19903]**{DRAFT} ⌈

| Kind: | type alias |
|---|---|
| Symbol: | Errc |
| Scope: | class ara::crypto::CryptoErrorDomain |
| Derived from: | CryptoErrc |
| Syntax: | `using ara::crypto::CryptoErrorDomain::Errc = CryptoErrc;` |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" |
| Description: | crypto error |

⌋*(RS_CRYPTO_02310)*

### [SWS_CRYPT_19904]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | type alias |
| **Symbol:** | Exception |
| **Scope:** | class ara::crypto::CryptoErrorDomain |
| **Derived from:** | CryptoException |
| **Syntax:** | `using ara::crypto::CryptoErrorDomain::Exception = CryptoException;` |
| **Header file:** | #include "ara/crypto/common/crypto_error_domain.h" |
| **Description:** | Alias for the exception base class. |

⌋*(RS_CRYPTO_02310)*

### [SWS_CRYPT_10018]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | enumeration | |
| **Symbol:** | KeySlotType | |
| **Scope:** | namespace ara::crypto | |
| **Underlying type:** | std::uint32_t | |
| **Syntax:** | `enum class KeySlotType :  std::uint32_t {...};` | |
| **Values:** | kMachine= 1 | machine type key-slot - can be managed by application |
| | kApplication= 2 | application exclusive type key-slot |
| **Header file:** | #include "ara/crypto/common/base_id_types.h" | |
| **Description:** | Enumeration of key-slot types; currently only machine and applicaiton key-slots are defined. | |

⌋*(RS_CRYPTO_02004)*

## 8.5   API Reference

### [SWS_CRYPT_10800]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | class |
| **Symbol:** | IOInterface |
| **Scope:** | namespace ara::crypto |
| **Syntax:** | `class ara::crypto::IOInterface {...};` |
| **Header file:** | #include "ara/crypto/common/io_interface.h" |
| **Description:** | Formal interface of an IOInterface is used for saving and loading of security objects. Actual saving and loading should be implemented by internal methods known to a trusted pair of Crypto Provider and Storage Provider. Each object should be uniquely identified by its type and Crypto Object Unique Identifier (COUID). This interface suppose that objects in the container are compressed i.e. have a minimal size optimized for. |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_10700]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | Serializable |
| Scope: | namespace ara::crypto |
| Syntax: | `class ara::crypto::Serializable {...};` |
| Header file: | #include "ara/crypto/common/serializable.h" |
| Description: | Serializable object interface. |

⌋*(RS_CRYPTO_02105)*

**[SWS_CRYPT_10850]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | VolatileTrustedContainer |
| Scope: | namespace ara::crypto |
| Syntax: | `class ara::crypto::VolatileTrustedContainer {...};` |
| Header file: | #include "ara/crypto/common/volatile_trusted_container.h" |
| Description: | This explicit interface of a volatile Trusted Container is used for buffering CryptoAPI objects in RAM. This class represents a "smart buffer" in that it provides access to the IOInterface, which can be used for querying meta-data of the buffer content. |

⌋*(RS_CRYPTO_02004)*

**[SWS_CRYPT_19905]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CryptoException |
| Scope: | namespace ara::crypto |
| Base class: | ara::core::Exception |
| Syntax: | `class ara::crypto::CryptoException :  public ara::core::Exception {...};` |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" |
| Description: | Exception type thrown for CRYPTO errors. |

⌋*(RS_CRYPTO_02310)*

**[SWS_CRYPT_19900]**{DRAFT} ⌈

| Kind: | class |
|---|---|
| Symbol: | CryptoErrorDomain |
| Scope: | namespace ara::crypto |
| Base class: | ara::core::ErrorDomain |
| Syntax: | `class ara::crypto::CryptoErrorDomain final :  public ara::core::Error Domain {...};` |
| Unique ID: | 0x8000'0000'0000'0801 |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" |

▽

△

| Description: | Crypto Error Domain class that provides interfaces as defined by ara::core::ErrorDomain such as a name of the Crypto Error Domain or messages for each error code. This class represents an error domain responsible for all errors that may be reported by public APIs in ara::crypto namespace. . |

⌋*(RS_AP_00130)*

### [SWS_CRYPT_19951]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | MakeErrorCode(CryptoErrorDomain::Errc code, ara::core::ErrorDomain::SupportDataType data) |
| Scope: | namespace ara::crypto |
| Syntax: | `constexpr ara::core::ErrorCode ara::crypto::MakeErrorCode (CryptoError Domain::Errc code, ara::core::ErrorDomain::SupportDataType data) noexcept;` |
| Parameters (in): | code | an error code identifier from the CryptoErrc enumeration |
| | data | supplementary data for the error description |
| Return value: | ara::core::ErrorCode | an instance of ErrorCode created according the arguments |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" |
| Description: | Makes Error Code instances from the Crypto Error Domain. The returned ErrorCode instance always references to CryptoErrorDomain. |

⌋*(RS_CRYPTO_02310)*

### [SWS_CRYPT_20099]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | LoadCryptoProvider(const ara::core::InstanceSpecifier &iSpecify) |
| Scope: | namespace ara::crypto |
| Syntax: | `ara::crypto::cryp::CryptoProvider::Uptr ara::crypto::LoadCrypto Provider (const ara::core::InstanceSpecifier &iSpecify) noexcept;` |
| Parameters (in): | iSpecify | the globally unique identifier of required Crypto Provider |
| Return value: | ara::crypto::cryp::CryptoProvider::Uptr | unique smart pointer to loaded Crypto Provider |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/common/entry_point.h" |
| Description: | Factory that creates or return existing single instance of specific Crypto Provider. If (providerUid == nullptr) then platform default provider should be loaded. |

⌋*(RS_CRYPTO_02401, RS_CRYPTO_02301)*

### [SWS_CRYPT_30099]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | LoadKeyStorageProvider() |
| Scope: | namespace ara::crypto |

▽

△

| Syntax: | `ara::crypto::keys::KeyStorageProvider::Uptr ara::crypto::LoadKey StorageProvider () noexcept;` | |
|---|---|---|
| Return value: | ara::crypto::keys::KeyStorage Provider::Uptr | unique smart pointer to loaded Key Storage Provider |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kRuntimeFault | if the Key Storage Provider instance cannot be created |
| Header file: | #include "ara/crypto/common/entry_point.h" | |
| Description: | Factory that creates or return existing single instance of the Key Storage Provider. | |

⌋(*RS_CRYPTO_02109*, *RS_CRYPTO_02401*, *RS_CRYPTO_02301*)

## [SWS_CRYPT_40099]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | LoadX509Provider() | |
| Scope: | namespace ara::crypto | |
| Syntax: | `ara::crypto::x509::X509Provider::Uptr ara::crypto::LoadX509Provider () noexcept;` | |
| Return value: | ara::crypto::x509::X509Provider::Uptr | unique smart pointer to loaded X.509 Provider |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kRuntimeFault | if the X.509 Provider cannot be loaded |
| Header file: | #include "ara/crypto/common/entry_point.h" | |
| Description: | Factory that creates or return existing single instance of the X.509 Provider. X.509 Provider should use the default Crypto Provider for hashing and signature verification! Therefore when you load the X.509 Provider, in background it loads the default Crypto Provider too. | |

⌋(*RS_CRYPTO_02306*, *RS_CRYPTO_02301*)

## [SWS_CRYPT_30098]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GenerateRandomData(std::uint32_t count) | |
| Scope: | namespace ara::crypto | |
| Syntax: | `ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::GenerateRandomData (std::uint32_t count) noexcept;` | |
| Parameters (in): | count | number of random bytes to generate |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | a buffer filled with the generated random sequence |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kBusyResource | if the used RNG is currently out-of-entropy and therefore cannot provide the requested number of random bytes |
| Header file: | #include "ara/crypto/common/entry_point.h" | |
| Description: | Return an allocated buffer with a generated random sequence of the requested size. | |

⌋(*RS_CRYPTO_02206*)

**[SWS_CRYPT_20098]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | GetSecureCounter() |
| Scope: | namespace ara::crypto |
| Syntax: | `ara::core::Result<SecureCounter> ara::crypto::GetSecureCounter () noexcept;` |
| Return value: | ara::core::Result< SecureCounter > | a SecureCounter struct made up of the two unsigned 64 bit values (LSQW and MSQW) |
| Exception Safety: | noexcept |
| Thread Safety: | Thread-safe |
| Errors: | CryptoErrorDomain::kUnsupported | if the Secure Counter is unsupported by the Crypto Stack implementation on this Platform |
| | CryptoErrorDomain::kAccessViolation | if current Actor has no permission to call this routine |
| Header file: | #include "ara/crypto/common/entry_point.h" |
| Description: | Get current value of 128 bit Secure Counter supported by the Crypto Stack. Secure Counter is a non-rollover monotonic counter that ensures incrementation of its value for each following call. The Secure Counter is presented by two 64 bit components: Most Significant Quadword (MSQW) and Least Significant Quadword (LSQW). During normal operation of the Crypto Stack, the MSQW value is fixed (unchangeable) and only LSQW should be incremented. The LSQW counter can be implemented in the "low-power" (always-powered-up) domain of the main CPU, but the MSQW in the Flash/EEPROM storage. But the MSQW must be incremented if the LSQW reaches the maximum value of all ones. Also the MSQW must be incremented during reinitialisation of the whole Crypto Stack (e.g. if the "low-power" supply was interrupted by some reason). Permission to execute this routine is subject of Identity and Access Management control and may be restricted by application manifest! |

⌋*(RS_CRYPTO_02401)*

**[SWS_CRYPT_10112]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | HasEarlierVersionThan(const CryptoObjectUid &anotherId) |
| Scope: | struct ara::crypto::CryptoObjectUid |
| Syntax: | `constexpr bool ara::crypto::CryptoObjectUid::HasEarlierVersionThan (const CryptoObjectUid &anotherId) const noexcept;` |
| Parameters (in): | anotherId | another identifier for the comparison |
| Return value: | bool | true if this identifier was generated earlier than the anotherId |
| Exception Safety: | noexcept |
| Thread Safety: | Reentrant |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" |
| Description: | Check whether this identifier was generated earlier than the one provided by the argument. |

⌋*(RS_CRYPTO_02006)*

**[SWS_CRYPT_10113]**{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | HasLaterVersionThan(const CryptoObjectUid &anotherId) |
| Scope: | struct ara::crypto::CryptoObjectUid |

▽

△

| Syntax: | constexpr bool ara::crypto::CryptoObjectUid::HasLaterVersionThan (const CryptoObjectUid &anotherId) const noexcept; | |
|---|---|---|
| Parameters (in): | anotherId | another identifier for the comparison |
| Return value: | bool | true if this identifier was generated later than the anotherId |
| Exception Safety: | noexcept | |
| Thread Safety: | Reentrant | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Check whether this identifier was generated later than the one provided by the argument. | |

⌋(*RS_CRYPTO_02006*)

## [SWS_CRYPT_10111]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | HasSameSourceAs(const CryptoObjectUid &anotherId) | |
| Scope: | struct ara::crypto::CryptoObjectUid | |
| Syntax: | constexpr bool ara::crypto::CryptoObjectUid::HasSameSourceAs (const CryptoObjectUid &anotherId) const noexcept; | |
| Parameters (in): | anotherId | another identifier for the comparison |
| Return value: | bool | true if both identifiers has common source (identical value of the mGeneratorUid field) |
| Exception Safety: | noexcept | |
| Thread Safety: | Reentrant | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Check whether this identifier has a common source with the one provided by the argument. | |

⌋(*RS_CRYPTO_02006*)

## [SWS_CRYPT_10114]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsNil() | |
| Scope: | struct ara::crypto::CryptoObjectUid | |
| Syntax: | bool ara::crypto::CryptoObjectUid::IsNil () const noexcept; | |
| Return value: | bool | true if this identifier is "Nil" and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Reentrant | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Check whether this identifier is "Nil". | |

⌋(*RS_CRYPTO_02006*)

## [SWS_CRYPT_10115]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | SourceIsNil() | |
| Scope: | struct ara::crypto::CryptoObjectUid | |

▽

△

| Syntax: | `bool ara::crypto::CryptoObjectUid::SourceIsNil () const noexcept;` | |
|---|---|---|
| Return value: | bool | true if this identifier is "Nil" and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Reentrant | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Check whether this object's generator identifier is "Nil". | |

⌋*(RS_CRYPTO_02006)*

## [SWS_CRYPT_10810]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~IOInterface() |
| Scope: | class ara::crypto::IOInterface |
| Syntax: | `virtual ara::crypto::IOInterface::~IOInterface () noexcept=default;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/common/io_interface.h" |
| Description: | Destructor. |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_10819]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetAllowedUsage() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | `virtual AllowedUsageFlags ara::crypto::IOInterface::GetAllowedUsage () const noexcept=0;` | |
| Return value: | AllowedUsageFlags | allowed key/seed usage flags |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Return actual allowed key/seed usage flags defined by the key slot prototype for this "Actor" and current content of the container. Volatile containers don't have any prototyped restrictions, but can have restrictions defined at run-time for a current instance of object. A value returned by this method is bitwise AND of the common usage flags defined at run-time and the usage flags defined by the UserPermissions prototype for current "Actor". This method is especially useful for empty permanent prototyped containers. | |

⌋*(RS_CRYPTO_02008)*

## [SWS_CRYPT_10813]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetCapacity() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | `virtual std::size_t ara::crypto::IOInterface::GetCapacity () const noexcept=0;` | |
| Return value: | std::size_t | capacity of the underlying buffer of this IOInterface (in bytes) |

▽

△

| Exception Safety: | noexcept |
|---|---|
| Thread Safety: | Thread-safe |
| Header file: | #include "ara/crypto/common/io_interface.h" |
| Description: | Return capacity of the underlying resource. |

⌋(RS_CRYPTO_02110)

## [SWS_CRYPT_10812]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetCryptoObjectType() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | virtual CryptoObjectType ara::crypto::IOInterface::GetCryptoObjectType () const noexcept=0; | |
| Return value: | CryptoObjectType | the CryptoObjectType stored inside the referenced resource |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Return the CryptoObjectType of the object referenced by this IOInterface. | |

⌋(RS_CRYPTO_02110)

## [SWS_CRYPT_10811]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetObjectId() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | virtual CryptoObjectUid ara::crypto::IOInterface::GetObjectId () const noexcept=0; | |
| Return value: | CryptoObjectUid | type of the content stored in the container |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Return COUID of an object stored to this IOInterface. If the container is empty then this method returns CryptoObjectType::KUndefined. Unambiguous identification of a crypto object requires both components: CryptoObjectUid and CryptoObjectType. | |

⌋(RS_CRYPTO_02004)

## [SWS_CRYPT_10817]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetPayloadSize() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | virtual std::size_t ara::crypto::IOInterface::GetPayloadSize () const noexcept=0; | |
| Return value: | std::size_t | size of an object payload stored in the underlying buffer of this IOInterface (in bytes) |

▽

△

| | |
|---|---|
| **Exception Safety:** | noexcept |
| **Thread Safety:** | Thread-safe |
| **Header file:** | #include "ara/crypto/common/io_interface.h" |
| **Description:** | Return size of an object payload stored in the underlying buffer of this IOInterface. If the container is empty then this method returns 0. Returned value does not take into account the object's meta-information properties, but their size is fixed and common for all crypto objects independently from their actual type. space for an object's meta-information automatically, according to their implementation details. |

⌋*(RS_CRYPTO_02109)*

## [SWS_CRYPT_10822]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | GetPrimitiveId() | |
| **Scope:** | class ara::crypto::IOInterface | |
| **Syntax:** | `virtual CryptoAlgId ara::crypto::IOInterface::GetPrimitiveId () const noexcept=0;` | |
| **Return value:** | CryptoAlgId | the binary Crypto Primitive ID |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/common/io_interface.h" | |
| **Description:** | Get vendor specific ID of the primitive. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_10818]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | GetTypeRestriction() | |
| **Scope:** | class ara::crypto::IOInterface | |
| **Syntax:** | `virtual CryptoObjectType ara::crypto::IOInterface::GetTypeRestriction () const noexcept=0;` | |
| **Return value:** | CryptoObjectType | an object type of allowed content (CryptoObject Type::kUndefined means without restriction) |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/common/io_interface.h" | |
| **Description:** | Return content type restriction of this IOInterface. If KeySlotPrototypeProps::mAllowContent TypeChange==TRUE, then kUndefined shall be returned. If a container has a type restriction different from CryptoObjectType::kUndefined then only objects of the mentioned type can be saved to this container. Volatile containers don't have any content type restrictions. | |

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02110)*

## [SWS_CRYPT_10816]{DRAFT} ⌈

| | |
|---|---|
| **Kind:** | function |
| **Symbol:** | IsObjectExportable() |
| **Scope:** | class ara::crypto::IOInterface |

▽

△

| Syntax: | virtual bool ara::crypto::IOInterface::IsObjectExportable () const noexcept=0; | |
|---|---|---|
| Return value: | bool | true if an object stored to the container has set the "exportable" attribute |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Return the "exportable" attribute of an object stored to the container. The exportability of an object doesn't depend from the volatility of its container. | |

⌋*(RS_CRYPTO_02109)*

## [SWS_CRYPT_10815]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsObjectSession() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | virtual bool ara::crypto::IOInterface::IsObjectSession () const noexcept=0; | |
| Return value: | bool | true if the object referenced by this IOInterface has set the "session" attribute |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Return the "session" (or "temporary") attribute of an object as set e.g. by KeyDerivation FunctionCtx::DeriveKey(). A "session" object can be stored to a VolatileTrustedContainer only! If this IOInterface is linked to a KeySlot this returns always false. | |

⌋*(RS_CRYPTO_02109)*

## [SWS_CRYPT_10814]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsVolatile() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | virtual bool ara::crypto::IOInterface::IsVolatile () const noexcept=0; | |
| Return value: | bool | true if the container has a volatile nature (i.e. "temporary" or "in RAM") or false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Return volatility of the the underlying buffer of this IOInterface. A "session" object can be stored to a "volatile" container only. A content of a "volatile" container will be destroyed together with the interface instance. | |

⌋*(RS_CRYPTO_02109)*

## [SWS_CRYPT_10823]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsValid() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | `virtual bool ara::crypto::IOInterface::IsValid () const noexcept=0;` | |
| Return value: | bool | true if the underlying resource can be valid, false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Get whether the underlying KeySlot is valid. An IOInterface is invalidated if the underlying resource has been modified after the IOInterface has been opened. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_10821]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsWritable() | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | `virtual bool ara::crypto::IOInterface::IsWritable () const noexcept=0;` | |
| Return value: | bool | true if the underlying resource can be written |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Get whether the underlying KeySlot is writable - if this IOInterface is linked to a VolatileTrusted Container always return true. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30202]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const IOInterface &other) | |
| Scope: | class ara::crypto::IOInterface | |
| Syntax: | `IOInterface& ara::crypto::IOInterface::operator= (const IOInterface &other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | IOInterface & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Copy-assign another IOInterface to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30203]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator=(IOInterface &&other) |
| Scope: | class ara::crypto::IOInterface |

▽

△

| Syntax: | `IOInterface& ara::crypto::IOInterface::operator= (IOInterface &&other)=default;` | |
|---|---|---|
| Parameters (in): | other | the other instance |
| Return value: | IOInterface & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/common/io_interface.h" | |
| Description: | Move-assign another IOInterface to this instance. | |

⌟(*RS_CRYPTO_02004*)

## [SWS_CRYPT_10150]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator==(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) | |
| Scope: | namespace ara::crypto | |
| Syntax: | `constexpr bool ara::crypto::operator== (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;` | |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if all members' values of lhs is equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Comparison operator "equal" for CryptoObjectUid operands. | |

⌟(*RS_CRYPTO_02005*)

## [SWS_CRYPT_10151]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator<(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) | |
| Scope: | namespace ara::crypto | |
| Syntax: | `constexpr bool ara::crypto::operator< (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;` | |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if a binary representation of lhs is less than rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Comparison operator "less than" for CryptoObjectUid operands. | |

⌟(*RS_CRYPTO_02005*)

### [SWS_CRYPT_10152]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator>(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) |
| Scope: | namespace ara::crypto |
| Syntax: | `constexpr bool ara::crypto::operator> (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;` |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if a binary representation of lhs is greater than rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Comparison operator "greater than" for CryptoObjectUid operands. | |

⌋*(RS_CRYPTO_02005)*

### [SWS_CRYPT_10153]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator!=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) |
| Scope: | namespace ara::crypto |
| Syntax: | `constexpr bool ara::crypto::operator!= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;` |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if at least one member of lhs has a value not equal to correspondent member of rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Comparison operator "not equal" for CryptoObjectUid operands. | |

⌋*(RS_CRYPTO_02005)*

### [SWS_CRYPT_10154]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator<=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) |
| Scope: | namespace ara::crypto |
| Syntax: | `constexpr bool ara::crypto::operator<= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;` |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if a binary representation of lhs is less than or equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |

▽

△

| Header file: | #include "ara/crypto/common/crypto_object_uid.h" |
|---|---|
| Description: | Comparison operator "less than or equal" for CryptoObjectUid operands. |

⌋(*RS_CRYPTO_02005*)

## [SWS_CRYPT_10155]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator>=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) |
| Scope: | namespace ara::crypto |
| Syntax: | `constexpr bool ara::crypto::operator>= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;` |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if a binary representation of lhs is greater than or equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/crypto_object_uid.h" | |
| Description: | Comparison operator "greater than or equal" for CryptoObjectUid operands. | |

⌋(*RS_CRYPTO_02005*)

## [SWS_CRYPT_10451]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator==(const Uuid &lhs, const Uuid &rhs) |
| Scope: | namespace ara::crypto |
| Syntax: | `constexpr bool ara::crypto::operator== (const Uuid &lhs, const Uuid &rhs) noexcept;` |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if a binary representation of lhs is equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/uuid.h" | |
| Description: | Comparison operator "equal" for Uuid operands. | |

⌋(*RS_CRYPTO_02112*)

## [SWS_CRYPT_10452]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | operator<(const Uuid &lhs, const Uuid &rhs) |
| Scope: | namespace ara::crypto |
| Syntax: | `constexpr bool ara::crypto::operator< (const Uuid &lhs, const Uuid &rhs) noexcept;` |
| Parameters (in): | lhs | left-hand side operand |

▽

△

| | rhs | right-hand side operand |
|---|---|---|
| **Return value:** | bool | true if a binary representation of lhs is less than rhs, and false otherwise |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/common/uuid.h" | |
| **Description:** | Comparison operator "less than" for Uuid operands. | |

⌋*(RS_CRYPTO_02112)*

## [SWS_CRYPT_10453]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | operator>(const Uuid &lhs, const Uuid &rhs) | |
| **Scope:** | namespace ara::crypto | |
| **Syntax:** | `constexpr bool ara::crypto::operator> (const Uuid &lhs, const Uuid &rhs) noexcept;` | |
| **Parameters (in):** | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| **Return value:** | bool | true if a binary representation of lhs is greater than rhs, and false otherwise |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/common/uuid.h" | |
| **Description:** | Comparison operator "greater than" for Uuid operands. | |

⌋*(RS_CRYPTO_02112)*

## [SWS_CRYPT_10454]{DRAFT} ⌈

| | | |
|---|---|---|
| **Kind:** | function | |
| **Symbol:** | operator!=(const Uuid &lhs, const Uuid &rhs) | |
| **Scope:** | namespace ara::crypto | |
| **Syntax:** | `constexpr bool ara::crypto::operator!= (const Uuid &lhs, const Uuid &rhs) noexcept;` | |
| **Parameters (in):** | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| **Return value:** | bool | true if a binary representation of lhs is not equal to rhs, and false otherwise |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | Thread-safe | |
| **Header file:** | #include "ara/crypto/common/uuid.h" | |
| **Description:** | Comparison operator "not equal" for Uuid operands. | |

⌋*(RS_CRYPTO_02112)*

### [SWS_CRYPT_10455]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator<=(const Uuid &lhs, const Uuid &rhs) | |
| Scope: | namespace ara::crypto | |
| Syntax: | `constexpr bool ara::crypto::operator<= (const Uuid &lhs, const Uuid &rhs) noexcept;` | |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if a binary representation of lhs is less than or equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/uuid.h" | |
| Description: | Comparison operator "less than or equal" for Uuid operands. | |

⌋*(RS_CRYPTO_02112)*

### [SWS_CRYPT_10456]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator>=(const Uuid &lhs, const Uuid &rhs) | |
| Scope: | namespace ara::crypto | |
| Syntax: | `constexpr bool ara::crypto::operator>= (const Uuid &lhs, const Uuid &rhs) noexcept;` | |
| Parameters (in): | lhs | left-hand side operand |
| | rhs | right-hand side operand |
| Return value: | bool | true if a binary representation of lhs is greater than or equal to rhs, and false otherwise |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Header file: | #include "ara/crypto/common/uuid.h" | |
| Description: | Comparison operator "greater than or equal" for Uuid operands. | |

⌋*(RS_CRYPTO_02112)*

### [SWS_CRYPT_19954]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ThrowAsException(const ara::core::ErrorCode &errorCode) | |
| Scope: | class ara::crypto::CryptoErrorDomain | |
| Syntax: | `void ara::crypto::CryptoErrorDomain::ThrowAsException (const ara::core::ErrorCode &errorCode) const override;` | |
| Parameters (in): | errorCode | an error code identifier from the CryptoErrc enumeration |
| Return value: | None | |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" | |
| Description: | throws exception of error code | |

⌋*(RS_CRYPTO_02310)*

### [SWS_CRYPT_19902]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | CryptoErrorDomain() |
| Scope: | class ara::crypto::CryptoErrorDomain |
| Syntax: | `constexpr ara::crypto::CryptoErrorDomain::CryptoErrorDomain () noexcept;` |
| Exception Safety: | noexcept |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" |
| Description: | Ctor of the CryptoErrorDomain. |

⌋*(RS_CRYPTO_02310)*

### [SWS_CRYPT_19950]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Name() | |
| Scope: | class ara::crypto::CryptoErrorDomain | |
| Syntax: | `const char* ara::crypto::CryptoErrorDomain::Name () const noexcept override;` | |
| Return value: | const char * | "Crypto" text |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" | |
| Description: | returns Text "Crypto" | |

⌋*(RS_CRYPTO_02310)*

### [SWS_CRYPT_19953]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | Message(ara::core::ErrorDomain::CodeType errorCode) | |
| Scope: | class ara::crypto::CryptoErrorDomain | |
| Syntax: | `const char* ara::crypto::CryptoErrorDomain::Message (ara::core::ErrorDomain::CodeType errorCode) const noexcept override;` | |
| Parameters (in): | errorCode | an error code identifier from the CryptoErrc enumeration |
| Return value: | const char * | message text of error code |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/common/crypto_error_domain.h" | |
| Description: | Translate an error code value into a text message. | |

⌋*(RS_CRYPTO_02310)*

### [SWS_CRYPT_10710]{DRAFT} ⌈

| Kind: | function |
|---|---|
| Symbol: | ~Serializable() |
| Scope: | class ara::crypto::Serializable |
| Syntax: | `virtual ara::crypto::Serializable::~Serializable () noexcept=default;` |
| Exception Safety: | noexcept |

▽

△

| Header file: | #include "ara/crypto/common/serializable.h" |
|---|---|
| Description: | Destructor. |

⌋(RS_CRYPTO_02004, RS_CRYPTO_02302)

## [SWS_CRYPT_10711]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ExportPublicly(FormatId formatId=kFormatDefault) | |
| Scope: | class ara::crypto::Serializable | |
| Syntax: | `virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ara::crypto::Serializable::ExportPublicly (FormatId formatId=kFormatDefault) const noexcept=0;` | |
| Parameters (in): | formatId | the Crypto Provider specific identifier of the output format |
| Return value: | ara::core::Result< ara::core::Vector< ara::core::Byte > > | a buffer with the serialized object |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if (output.empty() == false), but it's capacity is less than required |
| | CryptoErrorDomain::kUnknown Identifier | if an unknown format ID was specified |
| | CryptoErrorDomain::kUnsupported Format | if the specified format ID is not supported for this object type |
| Header file: | #include "ara/crypto/common/serializable.h" | |
| Description: | Serialize itself publicly. | |

⌋(RS_CRYPTO_02112)

## [SWS_CRYPT_10712]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | ExportPublicly(FormatId formatId=kFormatDefault) | |
| Scope: | class ara::crypto::Serializable | |
| Syntax: | `template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ara::crypto::Serializable::ExportPublicly (FormatId formatId=kFormatDefault) const noexcept;` | |
| Template param: | Alloc | custom allocator type of the output container |
| Parameters (in): | formatId | the Crypto Provider specific identifier of the output format |
| Return value: | ara::core::Result< ByteVector< Alloc > > | pre-reserved managed container for the serialization output |
| Exception Safety: | noexcept | |
| Thread Safety: | Thread-safe | |
| Errors: | CryptoErrorDomain::kInsufficient Capacity | if capacity of the output buffer is less than required |
| | CryptoErrorDomain::kUnknown Identifier | if an unknown format ID was specified |

▽

△

| | CryptoErrorDomain::kUnsupported Format | if the specified format ID is not supported for this object type |
|---|---|---|
| **Header file:** | #include "ara/crypto/common/serializable.h" | |
| **Description:** | Serialize itself publicly. This method sets the size of the output container according to actually saved value! | |

⌋*(RS_CRYPTO_02112)*

## [SWS_CRYPT_30204]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | operator=(const Serializable &other) | |
| **Scope:** | class ara::crypto::Serializable | |
| **Syntax:** | `Serializable& ara::crypto::Serializable::operator= (const Serializable &other)=default;` | |
| **Parameters (in):** | other | the other instance |
| **Return value:** | Serializable & | *this, containing the contents of other |
| **Header file:** | #include "ara/crypto/common/serializable.h" | |
| **Description:** | Copy-assign another Serializable to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_30205]{DRAFT} ⌈

| **Kind:** | function | |
|---|---|---|
| **Symbol:** | operator=(Serializable &&other) | |
| **Scope:** | class ara::crypto::Serializable | |
| **Syntax:** | `Serializable& ara::crypto::Serializable::operator= (Serializable &&other)=default;` | |
| **Parameters (in):** | other | the other instance |
| **Return value:** | Serializable & | *this, containing the contents of other |
| **Header file:** | #include "ara/crypto/common/serializable.h" | |
| **Description:** | Move-assign another Serializable to this instance. | |

⌋*(RS_CRYPTO_02004)*

## [SWS_CRYPT_10851]{DRAFT} ⌈

| **Kind:** | function |
|---|---|
| **Symbol:** | ~VolatileTrustedContainer() |
| **Scope:** | class ara::crypto::VolatileTrustedContainer |
| **Syntax:** | `virtual ara::crypto::VolatileTrustedContainer::~VolatileTrusted Container () noexcept=default;` |
| **Exception Safety:** | noexcept |
| **Header file:** | #include "ara/crypto/common/volatile_trusted_container.h" |
| **Description:** | Destructor. |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_10853]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | GetIOInterface() | |
| Scope: | class ara::crypto::VolatileTrustedContainer | |
| Syntax: | `virtual IOInterface& ara::crypto::VolatileTrustedContainer::Get`<br>`IOInterface () const noexcept=0;` | |
| Return value: | IOInterface & | a reference to the IOInterface of this container |
| Exception Safety: | noexcept | |
| Header file: | #include "ara/crypto/common/volatile_trusted_container.h" | |
| Description: | Retrieve the IOInterface used for importing/exporting objects into this container. | |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30206]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(const VolatileTrustedContainer &other) | |
| Scope: | class ara::crypto::VolatileTrustedContainer | |
| Syntax: | `VolatileTrustedContainer& ara::crypto::VolatileTrusted`<br>`Container::operator= (const VolatileTrustedContainer &other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | VolatileTrustedContainer & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/common/volatile_trusted_container.h" | |
| Description: | Copy-assign another VolatileTrustedContainer to this instance. | |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_30207]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | operator=(VolatileTrustedContainer &&other) | |
| Scope: | class ara::crypto::VolatileTrustedContainer | |
| Syntax: | `VolatileTrustedContainer& ara::crypto::VolatileTrusted`<br>`Container::operator= (VolatileTrustedContainer &&other)=default;` | |
| Parameters (in): | other | the other instance |
| Return value: | VolatileTrustedContainer & | *this, containing the contents of other |
| Header file: | #include "ara/crypto/common/volatile_trusted_container.h" | |
| Description: | Move-assign another VolatileTrustedContainer to this instance. | |

⌋*(RS_CRYPTO_02004)*

### [SWS_CRYPT_10411]{DRAFT} ⌈

| Kind: | function | |
|---|---|---|
| Symbol: | IsNil() | |
| Scope: | struct ara::crypto::Uuid | |
| Syntax: | `bool ara::crypto::Uuid::IsNil () const noexcept;` | |
| Return value: | bool | true if this identifier is "Nil" and false otherwise |
| Exception Safety: | noexcept | |

▽

△

| Thread Safety: | Thread-safe |
|---|---|
| Header file: | #include "ara/crypto/common/uuid.h" |
| Description: | Check whether this identifier is the "Nil UUID" (according to RFC4122). |

⌋(*RS_CRYPTO_02005*)

## [SWS_CRYPT_13000]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAlgIdUndefined |
| Scope: | namespace ara::crypto |
| Type: | const CryptoAlgId |
| Syntax: | `const CryptoAlgId ara::crypto::kAlgIdUndefined = 0u;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | Algorithm ID is undefined. Also this value may be used in meanings: Any or Default algorithm, None of algorithms. |
| | Effective values of Crypto Algorithm IDs are specific for concrete Crypto Stack implementation. But the zero value is reserved for especial purposes, that can differ depending from a usage context. This group defines a few constant names of the single zero value, but semantically they have different meaning specific for concrete application of the constant. |

⌋(*RS_CRYPTO_02107*)

## [SWS_CRYPT_13001]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAlgIdAny |
| Scope: | namespace ara::crypto |
| Type: | const CryptoAlgId |
| Syntax: | `const CryptoAlgId ara::crypto::kAlgIdAny = kAlgIdUndefined;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | Any Algorithm ID is allowed. |

⌋(*RS_CRYPTO_02107*)

## [SWS_CRYPT_13002]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAlgIdDefault |
| Scope: | namespace ara::crypto |
| Type: | const CryptoAlgId |
| Syntax: | `const CryptoAlgId ara::crypto::kAlgIdDefault = kAlgIdUndefined;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | Default Algorithm ID (in current context/primitive). |

⌋(*RS_CRYPTO_02107*)

**[SWS_CRYPT_13003]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAlgIdNone |
| Scope: | namespace ara::crypto |
| Type: | const CryptoAlgId |
| Syntax: | `const CryptoAlgId ara::crypto::kAlgIdNone = kAlgIdUndefined;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | None of Algorithm ID (i.e. an algorithm definition is not applicable). |

⌋*(RS_CRYPTO_02107)*

**[SWS_CRYPT_13102]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowDataDecryption |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowDataDecryption = 0x0002;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The key/seed can be used for data decryption initialization (applicable to symmetric and asymmetric algorithms). |

⌋*(RS_CRYPTO_02111)*

**[SWS_CRYPT_13101]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowDataEncryption |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowDataEncryption = 0x0001;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The key/seed can be used for data encryption initialization (applicable to symmetric and asymmetric algorithms). |

⌋*(RS_CRYPTO_02111)*

**[SWS_CRYPT_13113]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowDerivedDataDecryption |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowDerivedDataDecryption = kAllowDataDecryption << 16;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | A derived seed or symmetric key can be used for data decryption. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13112]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowDerivedDataEncryption |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowDerivedDataEncryption = kAllowDataEncryption << 16;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | A derived seed or symmetric key can be used for data encryption. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13117]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowDerivedRngInit |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowDerivedRngInit = kAllowRngInit << 16;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | A derived seed or symmetric key can be used for seeding of a RandomGeneratorContext. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13121]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowDerivedExactModeOnly |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowDerivedExactModeOnly = kAllowExactModeOnly << 16;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | Restrict usage of derived objects to specified operation mode only. A derived seed or symmetric key can be used only for the mode directly specified by Key::AlgId. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13118]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowDerivedKdfMaterial |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowDerivedKdfMaterial = kAllowKdfMaterial << 16;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | A derived seed or symmetric key can be used as a RestrictedUseObject for slave-keys derivation via a Key Derivation Function (KDF). |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13122]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowKdfMaterialAnyUsage |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowKdfMaterialAnyUsage = kAllowKdfMaterial | kAllowDerivedDataEncryption | kAllowDerivedDataDecryption | kAllowDerivedSignature | kAllowDerivedVerification | kAllowDerivedKeyDiversify | kAllowDerivedRngInit | kAllowDerivedKdfMaterial | kAllowDerivedKeyExporting | kAllowDerivedKeyImporting;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | Allow usage of the object as a key material for KDF and any usage of derived objects. The seed or symmetric key can be used as a RestrictedUseObject for a Key Derivation Function (KDF) and the derived "slave" keys can be used without limitations. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13116]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowDerivedKeyDiversify |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowDerivedKeyDiversify = kAllowKeyDiversify << 16;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | A derived seed or symmetric key can be used for slave-keys diversification. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13119]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowDerivedKeyExporting |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowDerivedKeyExporting = kAllowKeyExporting << 16;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | A derived seed or symmetric key can be used as a "transport" one for Key-Wrap transformation. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13120]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowDerivedKeyImporting |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |

▽

△

| | |
|---|---|
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowDerivedKeyImporting = k` `AllowKeyImporting << 16;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | A derived seed or symmetric key can be used as a "transport" one for Key-Unwrap transformation. |

⌊*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13114]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowDerivedSignature |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowDerivedSignature = kAllow` `Signature << 16;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | A derived seed or symmetric key can be used for MAC/HMAC production. |

⌊*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13115]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowDerivedVerification |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowDerivedVerification = k` `AllowVerification << 16;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | A derived seed or symmetric key can be used for MAC/HMAC verification. |

⌊*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13111]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowExactModeOnly |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowExactModeOnly = 0x8000;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | The key can be used only for the mode directly specified by Key::AlgId. |

⌊*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13108]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowKdfMaterial |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowKdfMaterial = 0x0080;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The object can be used as an input key material to KDF. The seed or symmetric key can be used as a RestrictedUseObject for slave-keys derivation via a Key Derivation Function (KDF). |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13105]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowKeyAgreement |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowKeyAgreement = 0x0010;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The seed or asymmetric key can be used for key-agreement protocol execution. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13106]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowKeyDiversify |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowKeyDiversify = 0x0020;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The seed or symmetric key can be used for slave-keys diversification. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13109]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowKeyExporting |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowKeyExporting = 0x0100;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The key can be used as "transport" one for Key-Wrap or Encapsulate transformations (applicable to symmetric and asymmetric keys). |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13110]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowKeyImporting |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowKeyImporting = 0x0200;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The key can be used as "transport" one for Key-Unwrap or Decapsulate transformations (applicable to symmetric and asymmetric keys). |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13100]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowPrototypedOnly |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowPrototypedOnly = 0;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | This group contains list of constant 1-bit values predefined for Allowed Usage flags. The key/seed usage will be fully specified by a key slot prototype (the object can be used only after reloading from the slot). |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13107]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowRngInit |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowRngInit = 0x0040;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The seed or symmetric key can be used for seeding of a RandomGeneratorCtx. |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13103]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kAllowSignature |
| Scope: | namespace ara::crypto |
| Type: | const AllowedUsageFlags |
| Syntax: | `const AllowedUsageFlags ara::crypto::kAllowSignature = 0x0004;` |
| Header file: | #include "ara/crypto/common/base_id_types.h" |
| Description: | The key/seed can be used for digital signature or MAC/HMAC production (applicable to symmetric and asymmetric algorithms). |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_13104]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | kAllowVerification |
| *Scope:* | namespace ara::crypto |
| *Type:* | const AllowedUsageFlags |
| *Syntax:* | `const AllowedUsageFlags ara::crypto::kAllowVerification = 0x0008;` |
| *Header file:* | #include "ara/crypto/common/base_id_types.h" |
| *Description:* | The key/seed can be used for digital signature or MAC/HMAC verification (applicable to symmetric and asymmetric algorithms). |

⌋*(RS_CRYPTO_02111)*

### [SWS_CRYPT_10102]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | mVersionStamp |
| *Scope:* | struct ara::crypto::CryptoObjectUid |
| *Type:* | std::uint64_t |
| *Syntax:* | `std::uint64_t ara::crypto::CryptoObjectUid::mVersionStamp = 0u;` |
| *Header file:* | #include "ara/crypto/common/crypto_object_uid.h" |
| *Description:* | Sequential value of a steady timer or simple counter, representing version of correspondent Crypto Object. |

⌋*(RS_CRYPTO_02006)*

### [SWS_CRYPT_30002]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | mLSQW |
| *Scope:* | struct ara::crypto::SecureCounter |
| *Type:* | std::uint64_t |
| *Syntax:* | `std::uint64_t ara::crypto::SecureCounter::mLSQW;` |
| *Header file:* | #include "ara/crypto/common/entry_point.h" |
| *Description:* | least significant 64 bits |

⌋*(RS_CRYPTO_02401)*

### [SWS_CRYPT_30003]{DRAFT} ⌈

| | |
|---|---|
| *Kind:* | variable |
| *Symbol:* | mMSQW |
| *Scope:* | struct ara::crypto::SecureCounter |
| *Type:* | std::uint64_t |
| *Syntax:* | `std::uint64_t ara::crypto::SecureCounter::mMSQW;` |
| *Header file:* | #include "ara/crypto/common/entry_point.h" |
| *Description:* | most significant 64 bits |

⌋*(RS_CRYPTO_02401)*

### [SWS_CRYPT_10750]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kFormatDefault |
| Scope: | class ara::crypto::Serializable |
| Type: | const FormatId |
| Syntax: | static const FormatId ara::crypto::Serializable::kFormatDefault = 0; |
| Header file: | #include "ara/crypto/common/serializable.h" |
| Description: | Default serialization format. |

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02302)*

### [SWS_CRYPT_10752]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kFormatDerEncoded |
| Scope: | class ara::crypto::Serializable |
| Type: | const FormatId |
| Syntax: | static const FormatId ara::crypto::Serializable::kFormatDerEncoded = 2; |
| Header file: | #include "ara/crypto/common/serializable.h" |
| Description: | Export DER-encoded value of an object. |

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02302)*

### [SWS_CRYPT_10753]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kFormatPemEncoded |
| Scope: | class ara::crypto::Serializable |
| Type: | const FormatId |
| Syntax: | static const FormatId ara::crypto::Serializable::kFormatPemEncoded = 3; |
| Header file: | #include "ara/crypto/common/serializable.h" |
| Description: | Export PEM-encoded value of an object. |

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02302)*

### [SWS_CRYPT_10751]{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | kFormatRawValueOnly |
| Scope: | class ara::crypto::Serializable |
| Type: | const FormatId |
| Syntax: | static const FormatId ara::crypto::Serializable::kFormatRawValueOnly = 1; |
| Header file: | #include "ara/crypto/common/serializable.h" |
| Description: | Export only raw value of an object. |

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02302)*

**[SWS_CRYPT_10412]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mQwordLs |
| Scope: | struct ara::crypto::Uuid |
| Type: | std::uint64_t |
| Syntax: | `std::uint64_t ara::crypto::Uuid::mQwordLs = 0u;` |
| Header file: | #include "ara/crypto/common/uuid.h" |
| Description: | Less significant QWORD. |

⌋*(RS_CRYPTO_02005)*

**[SWS_CRYPT_10413]**{DRAFT} ⌈

| Kind: | variable |
|---|---|
| Symbol: | mQwordMs |
| Scope: | struct ara::crypto::Uuid |
| Type: | std::uint64_t |
| Syntax: | `std::uint64_t ara::crypto::Uuid::mQwordMs = 0u;` |
| Header file: | #include "ara/crypto/common/uuid.h" |
| Description: | Most significant QWORD. |

⌋*(RS_CRYPTO_02005)*

# 9 Service Interfaces

No content defined.

## 9.1 Type definitions

No types are defined for service interfaces.

## 9.2 Provided Service Interfaces

No service interfaces are provided.

## 9.3 Required Service Interfaces

No service interfaces are required.

## 9.4 Application Errors

No application errors are defined.

# A   Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Chapter is generated.

| Class | AdaptiveApplicationSwComponentType | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure | | | |
| Note | This meta-class represents the ability to support the formal modeling of application software on the AUTOSAR adaptive platform. Consequently, it shall only be used on the AUTOSAR adaptive platform.<br><br>**Tags:**atp.recommendedPackage=AdaptiveApplicationSwComponentTypes | | | |
| Base | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType* | | | |
| Aggregated by | ARPackage.element | | | |
| Attribute | Type | Mult. | Kind | Note |
| internalBehavior | AdaptiveSwcInternal Behavior | 0..1 | aggr | This aggregation represents the internal behavior of the AdaptiveApplicationSwComponentType for the AUTOSAR adaptive platform.<br><br>**Stereotypes:** atpSplitable; atpVariation<br>**Tags:**<br>atp.Splitkey=internalBehavior.shortName, internal Behavior.variationPoint.shortLabel<br>vh.latestBindingTime=preCompileTime |

**Table A.1: AdaptiveApplicationSwComponentType**

| Class | CryptoCertificate | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment | | | |
| Note | This meta-class represents the ability to model a cryptographic certificate. | | | |
| Base | *ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| Aggregated by | CryptoModuleInstantiation.cryptoCertificate | | | |
| Attribute | Type | Mult. | Kind | Note |
| isPrivate | Boolean | 0..1 | attr | This attribute controls the possibility to access the content of the CryptoCertificateSlot by Find() interfaces of the X509 Provider. |

**Table A.2: CryptoCertificate**

| Class | CryptoCertificateInterface | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign | | | |
| Note | This meta-class provides the ability to define a PortInterface for a CryptoCertificate.<br>**Tags:**<br>atp.Status=candidate<br>atp.recommendedPackage=CryptoInterfaces | | | |
| Base | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *CryptoInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, PortInterface, *Referrable* | | | |
| Aggregated by | ARPackage.element | | | |
| Attribute | Type | Mult. | Kind | Note |

▽

△

| Class | CryptoCertificateInterface | | | |
|-------|------------|---|---|---|
| isPrivate | Boolean | 0..1 | attr | This attribute controls the possibility to access the content of the CryptoCertificateSlot by Find() interfaces of the X509 Provider.<br><br>**Tags:**atp.Status=candidate |
| writeAccess | Boolean | 0..1 | attr | This attribute defines whether the application has write-access to the CryptoCertificate (True) or only read-access (False).<br><br>**Tags:**atp.Status=candidate |

**Table A.3: CryptoCertificateInterface**

| Class | CryptoCertificateToCryptoKeySlotMapping | | | |
|-------|------------|---|---|---|
| *Package* | M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment | | | |
| *Note* | This meta-class represents the ability to define a mapping between a CryptoKeySlot and a Crypto Certificate. | | | |
| *Base* | *ARObject* | | | |
| *Aggregated by* | CryptoModuleInstantiation.certificateToKeySlotMapping | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| crypto Certificate | CryptoCertificate | 0..1 | ref | This reference represents the mapped cryptoCertificate. |
| cryptoKeySlot | CryptoKeySlot | 0..2 | ref | This reference represents the mapped cryptoKeySlot. |

**Table A.4: CryptoCertificateToCryptoKeySlotMapping**

| Class | CryptoKeySlot | | | |
|-------|------------|---|---|---|
| *Package* | M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment | | | |
| *Note* | This meta-class represents the ability to define a concrete key to be used for a crypto operation.<br><br>**Tags:**atp.ManifestKind=MachineManifest | | | |
| *Base* | *ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| *Aggregated by* | CryptoProvider.keySlot | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| allocateShadow Copy | Boolean | 0..1 | attr | This attribute defines whether a shadow copy of this Key Slot shall be allocated to enable rollback of a failed Key Slot update campaign (see interface BeginTransaction). |
| cryptoAlgId | String | 0..1 | attr | This attribute defines a crypto algorithm restriction (kAlgId Any means without restriction). The algorithm can be specified partially: family & length, mode, padding.<br><br>Future Crypto Providers can support some crypto algorithms that are not well known/ standardized today, therefore AUTOSAR doesn't provide a concrete list of crypto algorithms' identifiers and doesn't suppose usage of numerical identifiers. Instead of this a provider supplier should provide string names of supported algorithms in accompanying documentation. The name of a crypto algorithm shall follow the rules defined in the specification of cryptography for Adaptive Platform. |
| cryptoObject Type | CryptoObjectTypeEnum | 0..1 | attr | Object type that can be stored in the slot. If this field contains "Undefined" then mSlotCapacity must be provided and larger then 0.<br><br>**Tags:**atp.Status=candidate |

▽

△

| Class | CryptoKeySlot | | | |
|---|---|---|---|---|
| keySlotAllowed Modification | CryptoKeySlotAllowed Modification | 0..1 | aggr | Restricts how this keySlot may be used<br>**Tags:**atp.Status=candidate |
| keySlotContent AllowedUsage | CryptoKeySlotContent AllowedUsage | * | aggr | Restriction of allowed usage of a key stored to the slot.<br>**Tags:**atp.Status=candidate |
| slotCapacity | PositiveInteger | 0..1 | attr | Capacity of the slot in bytes to be reserved by the stack vendor. One use case is to define this value in case that the cryptoObjectType is undefined and the slot size can not be deduced from cryptoObjectType and cryptoAlgId. "0" means slot size can be deduced from cryptoObject Type and cryptoAlgId. |
| slotType | CryptoKeySlotType Enum | 0..1 | attr | This attribute defines whether the keySlot is exclusively used by the Application; or whether it is used by Stack Services and managed by a Key Manager Application.<br>**Tags:**atp.Status=candidate |

**Table A.5: CryptoKeySlot**

| Class | CryptoKeySlotInterface | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign | | | |
| **Note** | This meta-class provides the ability to define a PortInterface for Crypto Key Slots.<br>**Tags:**<br>atp.Status=candidate<br>atp.recommendedPackage=CryptoInterfaces | | | |
| **Base** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *CryptoInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| **Aggregated by** | ARPackage.element | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| allocateShadow Copy | Boolean | 0..1 | attr | This attribute defines whether a shadow copy of this Key Slot shall be allocated to enable rollback of a failed Key Slot update campaign (see interface BeginTransaction).<br>**Tags:**atp.Status=candidate |
| cryptoAlgId | String | 0..1 | attr | This attribute defines a crypto algorithm restriction (kAlgId Any means without restriction). The algorithm can be specified partially: family & length, mode, padding.<br><br>Future Crypto Providers can support some crypto algorithms that are not well known/ standardized today, therefore AUTOSAR doesn't provide a concrete list of crypto algorithms' identifiers and doesn't suppose usage of numerical identifiers. Instead of this a provider supplier should provide string names of supported algorithms in accompanying documentation. The name of a crypto algorithm shall follow the rules defined in the specification of cryptography for Adaptive Platform.<br>**Tags:**atp.Status=candidate |
| cryptoObject Type | CryptoObjectTypeEnum | 0..1 | attr | Object type that can be stored in the slot. If this field contains "Undefined" then mSlotCapacity must be provided and larger then 0<br>**Tags:**atp.Status=candidate |
| keySlotAllowed Modification | CryptoKeySlotAllowed Modification | 0..1 | aggr | Restricts how this keySlot may be used<br>**Tags:**atp.Status=candidate |

▽

△

| Class | CryptoKeySlotInterface | | | |
|---|---|---|---|---|
| keySlotContent AllowedUsage | CryptoKeySlotContent AllowedUsage | * | aggr | Restriction of allowed usage of a key stored to the slot. **Tags:**atp.Status=candidate |
| slotCapacity | PositiveInteger | 0..1 | attr | Capacity of the slot in bytes to be reserved by the stack vendor. One use case is to define this value in case that the cryptoObjectType is undefined and the slot size can not be deduced from cryptoObjectType and cryptoAlgId. "0" means slot size can be deduced from cryptoObject Type and cryptoAlgId. **Tags:**atp.Status=candidate |
| slotType | CryptoKeySlotType Enum | 0..1 | attr | This attribute defines whether the keySlot is exclusively used by the Application; or whether it is used by Stack Services and managed by a Key Manager Application. **Tags:**atp.Status=candidate |

**Table A.6: CryptoKeySlotInterface**

| Class | CryptoKeySlotToPortPrototypeMapping | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment | | | |
| **Note** | This meta-class represents the ability to define a mapping between a CryptoKeySlot on deployment level to a given PortPrototype that is typed by a CryptoKeySlotInterface. **Tags:**atp.recommendedPackage=CryptoKeySlotToPortPrototypeMappings | | | |
| **Base** | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement* | | | |
| **Aggregated by** | ARPackage.element | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| keySlot | CryptoKeySlot | 0..1 | ref | This reference represents the mapped CryptoKeySlot. |
| portPrototype | RPortPrototype | 0..1 | iref | This reference represents the mapped PortPrototype. **InstanceRef implemented by:**RPortPrototypeIn ExecutableInstanceRef |
| process | Process | 0..1 | ref | This reference represents the process required as context for the mapping. |

**Table A.7: CryptoKeySlotToPortPrototypeMapping**

| Class | CryptoProvider | | | |
|---|---|---|---|---|
| **Package** | M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment | | | |
| **Note** | CryptoProvider implements cryptographic primitives (algorithms) supported by the stack. Implementation of this component may be software or hardware based (HSM/TPM). | | | |
| **Base** | *ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| **Aggregated by** | CryptoModuleInstantiation.cryptoProvider | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| cryptoProvider Documentation | Documentation | 0..1 | ref | Documentation of the CryptoProvider that describes the implemented cryptographic primitives. |
| keySlot | CryptoKeySlot | * | aggr | This aggregation represents the key slots that are allocated by the CryptoProvider. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=keySlot.shortName |

**Table A.8: CryptoProvider**

| Class | CryptoProviderInterface |
|---|---|
| **Package** | M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign |
| **Note** | This meta-class provides the ability to define a PortInterface for a CryptoProvider.<br><br>**Tags:**<br>atp.Status=candidate<br>atp.recommendedPackage=CryptoInterfaces |
| **Base** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *CryptoInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, PortInterface, *Referrable* |
| **Aggregated by** | ARPackage.element |

| Attribute | Type | Mult. | Kind | Note |
|---|---|---|---|---|
| – | – | – | – | – |

**Table A.9: CryptoProviderInterface**

| Class | CryptoProviderToPortPrototypeMapping |
|---|---|
| **Package** | M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment |
| **Note** | This meta-class represents the ability to define a mapping between a CryptoProvider on deployment level to a given PortPrototype that is typed by a CryptoProviderInterface.<br><br>**Tags:**atp.recommendedPackage=CryptoProviderToPortPrototypeMappings |
| **Base** | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement* |
| **Aggregated by** | ARPackage.element |

| Attribute | Type | Mult. | Kind | Note |
|---|---|---|---|---|
| cryptoProvider | CryptoProvider | 0..1 | ref | This reference represents the mapped cryptoProvider. |
| portPrototype | RPortPrototype | 0..1 | iref | This reference represents the mapped PortPrototype.<br><br>**InstanceRef implemented by:**RPortPrototypeIn ExecutableInstanceRef |
| process | Process | 0..1 | ref | This reference represents the process required as context for the mapping. |

**Table A.10: CryptoProviderToPortPrototypeMapping**

| Class | CryptoServiceCertificate |
|---|---|
| **Package** | M2::AUTOSARTemplates::SystemTemplate::SecureCommunication |
| **Note** | This meta-class represents the ability to model a cryptographic certificate.<br><br>**Tags:**atp.recommendedPackage=CryptoServiceCertificates |
| **Base** | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable* |
| **Aggregated by** | ARPackage.element |

| Attribute | Type | Mult. | Kind | Note |
|---|---|---|---|---|
| algorithmFamily | CryptoCertificate AlgorithmFamilyEnum | 0..1 | attr | This attribute represents a description of the family of crypto algorithm used to generate public key and signature of the cryptographic certificate. |
| format | CryptoCertificateFormat Enum | 0..1 | attr | This attribute can be used to provide information about the format used to create the certificate |
| maximum Length | PositiveInteger | 0..1 | attr | This attribute represents the ability to define the maximum length of the certificate in bytes. |
| nextHigher Certificate | CryptoService Certificate | 0..1 | ref | The reference identifies the next higher certificate in the certificate chain. |

▽

△

| Class | CryptoServiceCertificate | | | |
|---|---|---|---|---|
| serverName Identification | String | 0..1 | attr | Server Name Indication (SNI) is needed if the IP address hosts multiple servers (on the same port), each of them using a different certificate. If the client sends the SNI to the Server in the client hello, the server looks the SNI up in its certificate list and uses the certificate identified by the SNI. |

**Table A.11: CryptoServiceCertificate**

| Class | PortInterface (abstract) | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::SWComponentTemplate::PortInterface | | | |
| Note | Abstract base class for an interface that is either provided or required by a port of a software component. | | | |
| Base | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| Subclasses | *AbstractRawDataStreamInterface*, *AbstractSynchronizedTimeBaseInterface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallStateSwitchInterface, LogAndTrace Interface, ModeSwitchInterface, *PersistencyInterface*, *PlatformHealthManagementInterface*, Security EventReportInterface, ServiceInterface, *StateManagementPortInterface*, TriggerInterface | | | |
| Aggregated by | ARPackage.element | | | |
| Attribute | Type | Mult. | Kind | Note |
| namespace (ordered) | SymbolProps | * | aggr | This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=namespace.shortName atp.Status=draft |

**Table A.12: PortInterface**

| Class | Process | | | |
|---|---|---|---|---|
| Package | M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest | | | |
| Note | This meta-class provides information required to execute the referenced executable. **Tags:**atp.recommendedPackage=Processes | | | |
| Base | *ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement* | | | |
| Aggregated by | ARPackage.element | | | |
| Attribute | Type | Mult. | Kind | Note |
| design | ProcessDesign | 0..1 | ref | This reference represents the identification of the design-time representation for the Process that owns the reference. |
| executable | Executable | * | ref | Reference to executable that is executed in the process. **Stereotypes:** atpUriDef |
| functionCluster Affiliation | String | 0..1 | attr | This attribute specifies which functional cluster the process is affiliated with. |
| numberOf RestartAttempts | PositiveInteger | 0..1 | attr | This attribute defines how often a process shall be restarted if the start fails. numberOfRestartAttempts = "0" OR Attribute not existing, start once numberOfRestartAttempts = "1", start a second time |

▽

△

| *Class* | **Process** | | | |
|---------|-------------|---|---|---|
| preMapping | Boolean | 0..1 | attr | This attribute describes whether the executable is preloaded into the memory. |
| processState Machine | ModeDeclarationGroup Prototype | 0..1 | aggr | Set of Process States that are defined for the process. |
| securityEvent | SecurityEventDefinition | * | ref | The reference identifies the collection of SecurityEvents that can be reported by the enclosing SoftwareCluster. **Stereotypes:** atpSplitable; atpUriDef **Tags:** atp.Splitkey=securityEvent atp.Status=candidate |
| stateDependent StartupConfig | StateDependentStartup Config | * | aggr | Applicable startup configurations. |

**Table A.13: Process**

| *Class* | **RPortPrototype** | | | |
|---------|-------------------|---|---|---|
| *Package* | M2::AUTOSARTemplates::SWComponentTemplate::Components | | | |
| *Note* | Component port requiring a certain port interface. | | | |
| *Base* | *ARObject*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable* | | | |
| *Aggregated by* | *AtpClassifier*.atpFeature, *SwComponentType*.port | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| required Interface | PortInterface | 0..1 | tref | The interface that this port requires. **Stereotypes:** isOfType |

**Table A.14: RPortPrototype**

# B Interfaces to other Functional Clusters (informative)

## B.1 Overview

AUTOSAR decided not to standardize interfaces which are exclusively used between `Functional Clusters` (on platform-level only), to allow efficient implementations, which might depend e.g. on the used Operating System.

This chapter provides informative guidelines how the interaction between `Functional Clusters` looks like, by clustering the relevant requirements of this document to describe Inter-Functional Cluster (IFC) interfaces. In addition, the standardized public interfaces which are accessible by user space applications (see chapters 8 and 9) can also be used for interaction between `Functional Clusters`.

The goal is to provide a clear understanding of `Functional Cluster` boundaries and interaction, without specifying syntactical details. This ensures compatibility between documents specifying different `Functional Clusters` and supports parallel implementation of different `Functional Clusters`. Details of the interfaces are up to the platform provider. Additional interfaces, parameters and return values can be added.

## B.2 Interface Tables

No content defined.