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	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Improve Crypto APIs by removing default values and using out parameter instead of returning ara::core::vector • Improve Keyslot and Certificate access control • Improve UpdatesObserver functional specification • Improve Signature handling • Add Log and Trace functionality to Crypto API
2023-11-23	R23-11	AUTOSAR Release Management	• Improved Crypto API Functional specification in Chapter 7 • Improved Chapter 8. API specification • Added initial version from Chapter 5. Dependencies to other functional clusters

▽

△

2022-11-24	R22-11	AUTOSAR Release Management	• Added: [SWS_CRYPT_40984] [SWS_CRYPT_40985] • Updated: [SWS_CRYPT_00502] [SWS_CRYPT_00503] [SWS_CRYPT_40983] [SWS_CRYPT_01820] [SWS_CRYPT_01821] [SWS_CRYPT_00919] [SWS_CRYPT_03303] [SWS_CRYPT_03305] [SWS_CRYPT_04204] [SWS_CRYPT_01821] [SWS_CRYPT_00919] • Updated API: [SWS_CRYPT_40981] [SWS_CRYPT_20746] [SWS_CRYPT_21115] [SWS_CRYPT_21312] [SWS_CRYPT_22711] [SWS_CRYPT_23012] [SWS_CRYPT_23013] [SWS_CRYPT_23014] • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Reworked SWS items in chapter 7 to improve testability • Improved traceability between chapters 7 and 8
2020-11-30	R20-11	AUTOSAR Release Management	• Rewrote the document to align with AUTOSAR standard • 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 • All bugs found after R18-03 are fixed • Crypto API is converted for usage of basic `ara::core` types • Crypto API is converted for support of the "Exception-less" approach • 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 • Added specification of additional Direct Crypto API (chapter 9)
2017-10-27	R17-10	AUTOSAR Release Management	• Initial release

Disclaimer

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_FO_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 11889 – ISO/IEC 11889-1:2015 Information technology - Trusted platform module library - Part 1: Architecture
https://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_AP_SWS_Core

[6] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[7] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

[8] Requirements on Cryptography
AUTOSAR_AP_RS_Cryptography

[9] Specification of Crypto Driver
AUTOSAR_CP_SWS_CryptoDriver

[10] PKCS #11 standard defines a platform-independent API to cryptographic tokens, such as hardware security modules (HSM)
https://www.oasis-open.org/standard/pkcs-11-specification-version-3-1/

[11] Platform Security Architecture (PSA)
https://arm-software.github.io/psa-api/

[12] 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]

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

[14] 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

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

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

[17] ISO IEC 9797-3:2011 Amd 1:2020(en) Information technology - Security techniques - Message Authentication Codes (MAC)
https://www.iso.org

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[34] Specification of Secure Hardware Extensions
AUTOSAR_FO_TR_SecureHardwareExtensions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[55] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[56] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

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

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 "Provided Interfaces" lists the interfaces provided by `Cryptography` to other Functional Clusters. Section 5.2 "Required Interfaces" lists the interfaces required by `Cryptography`.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [6].

5.1 Provided Interfaces

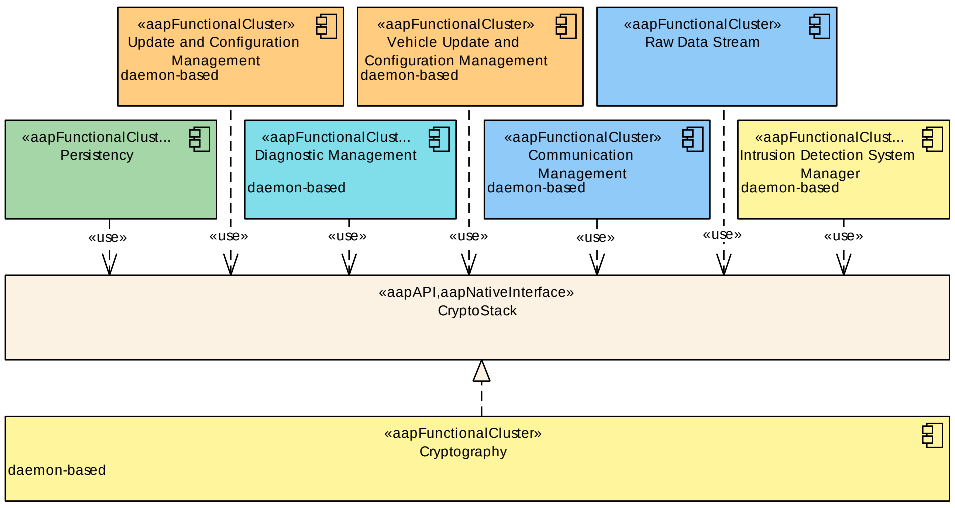

Figure 5.1: Interfaces provided by Cryptography to other Functional Clusters

Figure 5.1 shows interfaces assumed to be provided by `Cryptography` to other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.1 provides a list of interfaces assumed to be provided to other Functional Clusters within the AUTOSAR Adaptive Platform. Please note that for brevity only a placeholder interface `CryptoStack` is used instead of the many different standardized interfaces that would be required e.g., to access a certain key or to calculate a checksum.

Interface	***Functional Cluster***	***Purpose***
`CryptoStack`	Communication Management	This interface may be used e.g., to establish encrypted connections and generate / verify checksums (MAC).
	Diagnostic Management	This interface may be used e.g., to access keys for secure diagnostics.
	Intrusion Detection System Manager	`Adaptive Intrusion Detection System Manager` uses this interface to sign security events.
	Persistency	Used to ensure confidentiality and integrity of the persisted data.

▽

△

Interface	*Functional Cluster*	*Purpose*
	Raw Data Stream	This interface may be used to establish encrypted connections.
	Time Synchronization	`Time Synchronization` shall use this interface to generate / verify checksums (MAC).
	Update and Configuration Management	This interface may be used e.g., to verify the integrity and authenticity of `Software Package`s.
	Vehicle Update and Configuration Management	This interface may be used e.g., to verify the integrity and authenticity of `Vehicle Package`s.

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

Table 5.2 provides a list of interfaces assumed to be required from other Functional Clusters within the AUTOSAR Adaptive Platform.

Functional Cluster	*Interface*	*Purpose*
No required interfaces		

Table 5.2: Interfaces required from other Functional Clusters

6 Requirements Tracing

The following tables reference the requirements specified in [7] and [8] 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] [SWS_CRYPT_20746] [SWS_CRYPT_20747] [SWS_CRYPT_20748] [SWS_CRYPT_20750] [SWS_CRYPT_20751] [SWS_CRYPT_20752] [SWS_CRYPT_20753] [SWS_CRYPT_20754] [SWS_CRYPT_20755] [SWS_CRYPT_20756] [SWS_CRYPT_20757] [SWS_CRYPT_20758]
[RS_CRYPTO_02001]	The Crypto Stack shall conceal symmetric keys from the users	[SWS_CRYPT_00007] [SWS_CRYPT_20733] [SWS_CRYPT_20762] [SWS_CRYPT_20763] [SWS_CRYPT_20764] [SWS_CRYPT_20765] [SWS_CRYPT_20810] [SWS_CRYPT_21010] [SWS_CRYPT_21313] [SWS_CRYPT_21413] [SWS_CRYPT_21525] [SWS_CRYPT_21815] [SWS_CRYPT_22118] [SWS_CRYPT_22211] [SWS_CRYPT_22913] [SWS_CRYPT_23211] [SWS_CRYPT_23515] [SWS_CRYPT_23623] [SWS_CRYPT_23710] [SWS_CRYPT_23800] [SWS_CRYPT_23911] [SWS_CRYPT_24018] [SWS_CRYPT_24115] [SWS_CRYPT_41019]
[RS_CRYPTO_02002]	The Crypto Stack shall conceal asymmetric private keys from the users	[SWS_CRYPT_00007] [SWS_CRYPT_10305] [SWS_CRYPT_20733] [SWS_CRYPT_20762] [SWS_CRYPT_20763] [SWS_CRYPT_20764] [SWS_CRYPT_20765] [SWS_CRYPT_22500]
[RS_CRYPTO_02003]	The Crypto Stack shall support management of non-persistent session/ephemeral keys during their lifetime	[SWS_CRYPT_20512] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_20810] [SWS_CRYPT_21010] [SWS_CRYPT_21313] [SWS_CRYPT_21413] [SWS_CRYPT_21525] [SWS_CRYPT_21815] [SWS_CRYPT_22118] [SWS_CRYPT_22211] [SWS_CRYPT_22913] [SWS_CRYPT_23211] [SWS_CRYPT_23515] [SWS_CRYPT_23623] [SWS_CRYPT_23710] [SWS_CRYPT_23911] [SWS_CRYPT_24018] [SWS_CRYPT_24115] [SWS_CRYPT_41019]
[RS_CRYPTO_02004]	The Crypto Stack shall support secure storage of cryptographic artifacts	[SWS_CRYPT_00102] [SWS_CRYPT_00103] [SWS_CRYPT_04202] [SWS_CRYPT_04203] [SWS_CRYPT_04204] [SWS_CRYPT_04205] [SWS_CRYPT_04207] [SWS_CRYPT_04208] [SWS_CRYPT_04209] [SWS_CRYPT_10000] [SWS_CRYPT_10016] [SWS_CRYPT_10018] [SWS_CRYPT_10019] [SWS_CRYPT_10031] [SWS_CRYPT_10033] [SWS_CRYPT_10701] [SWS_CRYPT_10710] [SWS_CRYPT_10800] [SWS_CRYPT_10810] [SWS_CRYPT_10811] [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] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CRYPT_30123] [SWS_CRYPT_30124] [SWS_CRYPT_30125] [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] [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] [SWS_CRYPT_40995] [SWS_CRYPT_40996] [SWS_CRYPT_40997] [SWS_CRYPT_40998] [SWS_CRYPT_40999] [SWS_CRYPT_41000] [SWS_CRYPT_41001] [SWS_CRYPT_41002] [SWS_CRYPT_41003] [SWS_CRYPT_41004] [SWS_CRYPT_41005] [SWS_CRYPT_41006] [SWS_CRYPT_41007] [SWS_CRYPT_41008] [SWS_CRYPT_41009] [SWS_CRYPT_41010] [SWS_CRYPT_41011] [SWS_CRYPT_41012] [SWS_CRYPT_41013] [SWS_CRYPT_41014] [SWS_CRYPT_41015] [SWS_CRYPT_41016] [SWS_CRYPT_41017] [SWS_CRYPT_41018] [SWS_CRYPT_41028] [SWS_CRYPT_41029] [SWS_CRYPT_41031] [SWS_CRYPT_41032] [SWS_CRYPT_41036] [SWS_CRYPT_41037] [SWS_CRYPT_41038] [SWS_CRYPT_41053]
[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_30500] [SWS_CRYPT_41020]

▽

△

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] [SWS_CRYPT_10100] [SWS_CRYPT_10101] [SWS_CRYPT_10102] [SWS_CRYPT_10111] [SWS_CRYPT_10112] [SWS_CRYPT_10113] [SWS_CRYPT_10114] [SWS_CRYPT_10115] [SWS_CRYPT_20102] [SWS_CRYPT_20703] [SWS_CRYPT_20724] [SWS_CRYPT_20725] [SWS_CRYPT_20726] [SWS_CRYPT_20727] [SWS_CRYPT_20733] [SWS_CRYPT_20802] [SWS_CRYPT_21002] [SWS_CRYPT_21102] [SWS_CRYPT_21302] [SWS_CRYPT_21402] [SWS_CRYPT_21517] [SWS_CRYPT_21802] [SWS_CRYPT_22102] [SWS_CRYPT_22210] [SWS_CRYPT_22902] [SWS_CRYPT_23210] [SWS_CRYPT_23510] [SWS_CRYPT_23602] [SWS_CRYPT_23702] [SWS_CRYPT_24002] [SWS_CRYPT_24102] [SWS_CRYPT_40958] [SWS_CRYPT_41039] [SWS_CRYPT_41041]
[RS_CRYPTO_02007]	The Crypto Stack shall provide means for secure handling of "secret seeds"	[SWS_CRYPT_00102] [SWS_CRYPT_10401] [SWS_CRYPT_20723] [SWS_CRYPT_21311] [SWS_CRYPT_21411] [SWS_CRYPT_21516] [SWS_CRYPT_21810] [SWS_CRYPT_23000] [SWS_CRYPT_23001] [SWS_CRYPT_23002] [SWS_CRYPT_23003] [SWS_CRYPT_23011] [SWS_CRYPT_23012] [SWS_CRYPT_23013] [SWS_CRYPT_23014] [SWS_CRYPT_23015] [SWS_CRYPT_23016] [SWS_CRYPT_24015]
[RS_CRYPTO_02008]	The Crypto Stack shall support restrictions of the allowed usage scope for keys and "secret seeds"	[SWS_CRYPT_10004] [SWS_CRYPT_10819] [SWS_CRYPT_20400] [SWS_CRYPT_20401] [SWS_CRYPT_20402] [SWS_CRYPT_20411] [SWS_CRYPT_21521] [SWS_CRYPT_24800] [SWS_CRYPT_24801] [SWS_CRYPT_24811] [SWS_CRYPT_29046]
[RS_CRYPTO_02009]	The Crypto stack shall support separation of applications" access rights for each cryptographic object slot	[SWS_CRYPT_10003] [SWS_CRYPT_10004] [SWS_CRYPT_30208] [SWS_CRYPT_30300] [SWS_CRYPT_30405] [SWS_CRYPT_41025]
[RS_CRYPTO_02101]	The Crypto Stack shall provide interfaces to generate cryptographic keys for all supported primitives	[SWS_CRYPT_00601] [SWS_CRYPT_00603] [SWS_CRYPT_00608] [SWS_CRYPT_00609] [SWS_CRYPT_00610] [SWS_CRYPT_00611] [SWS_CRYPT_03300] [SWS_CRYPT_03311] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_40944] [SWS_CRYPT_40945] [SWS_CRYPT_40946] [SWS_CRYPT_40962] [SWS_CRYPT_40969]
[RS_CRYPTO_02102]	The Crypto Stack shall prevent keys from being used in incompatible or insecure ways	[SWS_CRYPT_00102] [SWS_CRYPT_03312] [SWS_CRYPT_10014] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21412] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21515] [SWS_CRYPT_21523] [SWS_CRYPT_21813] [SWS_CRYPT_41052]
[RS_CRYPTO_02103]	The Crypto Stack shall support primitives to derive cryptographic key material from a base key material	[SWS_CRYPT_03313] [SWS_CRYPT_10402] [SWS_CRYPT_20748] [SWS_CRYPT_21500] [SWS_CRYPT_21501] [SWS_CRYPT_21519] [SWS_CRYPT_21520] [SWS_CRYPT_21522]

▽

△

Requirement	Description	Satisfied by
[RS_CRYPTO_02104]	The Crypto Stack shall support a primitive to exchange cryptographic keys with another entity	[SWS_CRYPT_03301] [SWS_CRYPT_20743] [SWS_CRYPT_20752] [SWS_CRYPT_20753] [SWS_CRYPT_20758] [SWS_CRYPT_21300] [SWS_CRYPT_21301] [SWS_CRYPT_21400] [SWS_CRYPT_21401] [SWS_CRYPT_21800] [SWS_CRYPT_24000]
[RS_CRYPTO_02105]	Symmetric keys and asymmetric private keys shall be imported and exported in a secure format.	[SWS_CRYPT_03302] [SWS_CRYPT_03303] [SWS_CRYPT_03304] [SWS_CRYPT_04200] [SWS_CRYPT_10403] [SWS_CRYPT_10700] [SWS_CRYPT_20728] [SWS_CRYPT_20729] [SWS_CRYPT_20730] [SWS_CRYPT_20731] [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] [SWS_CRYPT_01506] [SWS_CRYPT_01508] [SWS_CRYPT_01651] [SWS_CRYPT_02123] [SWS_CRYPT_10014] [SWS_CRYPT_20710] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21515] [SWS_CRYPT_21523] [SWS_CRYPT_40964]
[RS_CRYPTO_02108]	The Crypto Stack shall provide interfaces for management and usage of algorithm-specific domain parameters	[SWS_CRYPT_20414] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21314] [SWS_CRYPT_21412] [SWS_CRYPT_21414] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21515] [SWS_CRYPT_21523] [SWS_CRYPT_21524] [SWS_CRYPT_21813] [SWS_CRYPT_21816] [SWS_CRYPT_22120] [SWS_CRYPT_22212] [SWS_CRYPT_22511] [SWS_CRYPT_23212] [SWS_CRYPT_23516] [SWS_CRYPT_23627] [SWS_CRYPT_23712] [SWS_CRYPT_24019] [SWS_CRYPT_24116] [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] [SWS_CRYPT_10801] [SWS_CRYPT_10802] [SWS_CRYPT_10814] [SWS_CRYPT_10815] [SWS_CRYPT_10816] [SWS_CRYPT_10817] [SWS_CRYPT_20701] [SWS_CRYPT_30099] [SWS_CRYPT_30100]
[RS_CRYPTO_02110]	The Crypto Stack shall support prototyping of application-exclusive key slot resources	[SWS_CRYPT_00101] [SWS_CRYPT_10812] [SWS_CRYPT_10813] [SWS_CRYPT_10818] [SWS_CRYPT_30300] [SWS_CRYPT_30301] [SWS_CRYPT_30302] [SWS_CRYPT_30305] [SWS_CRYPT_30306] [SWS_CRYPT_30307] [SWS_CRYPT_30308] [SWS_CRYPT_30309] [SWS_CRYPT_30310] [SWS_CRYPT_30311] [SWS_CRYPT_30312] [SWS_CRYPT_30313] [SWS_CRYPT_30350] [SWS_CRYPT_30351] [SWS_CRYPT_30407] [SWS_CRYPT_41042] [SWS_CRYPT_41043] [SWS_CRYPT_41044] [SWS_CRYPT_41045] [SWS_CRYPT_41046] [SWS_CRYPT_41047] [SWS_CRYPT_41048] [SWS_CRYPT_41049] [SWS_CRYPT_41051]

▽

△

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] [SWS_CRYPT_13100] [SWS_CRYPT_13101] [SWS_CRYPT_13102] [SWS_CRYPT_13103] [SWS_CRYPT_13104] [SWS_CRYPT_13105] [SWS_CRYPT_13106] [SWS_CRYPT_13107] [SWS_CRYPT_13108] [SWS_CRYPT_13109] [SWS_CRYPT_13110] [SWS_CRYPT_13111] [SWS_CRYPT_13112] [SWS_CRYPT_13113] [SWS_CRYPT_13114] [SWS_CRYPT_13115] [SWS_CRYPT_13116] [SWS_CRYPT_13117] [SWS_CRYPT_13118] [SWS_CRYPT_13119] [SWS_CRYPT_13120] [SWS_CRYPT_13121] [SWS_CRYPT_13122] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21515] [SWS_CRYPT_21523] [SWS_CRYPT_30500] [SWS_CRYPT_30501] [SWS_CRYPT_30503] [SWS_CRYPT_30505] [SWS_CRYPT_30506] [SWS_CRYPT_30508] [SWS_CRYPT_30510] [SWS_CRYPT_30511] [SWS_CRYPT_30550] [SWS_CRYPT_30551] [SWS_CRYPT_40991] [SWS_CRYPT_41024]
[RS_CRYPTO_02112]	The Crypto Stack shall execute export/import of a key value together with its meta information	[SWS_CRYPT_04200] [SWS_CRYPT_10200] [SWS_CRYPT_10451] [SWS_CRYPT_10452] [SWS_CRYPT_10453] [SWS_CRYPT_10454] [SWS_CRYPT_10455] [SWS_CRYPT_10456] [SWS_CRYPT_10711] [SWS_CRYPT_20005] [SWS_CRYPT_20728] [SWS_CRYPT_20729] [SWS_CRYPT_20730] [SWS_CRYPT_20731] [SWS_CRYPT_20732] [SWS_CRYPT_41026]
[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] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21312] [SWS_CRYPT_21315] [SWS_CRYPT_21412] [SWS_CRYPT_21515] [SWS_CRYPT_21523] [SWS_CRYPT_21813] [SWS_CRYPT_22511] [SWS_CRYPT_24016]
[RS_CRYPTO_02116]	The Crypto Stack shall support version control of key objects kept in the Key Storage	[SWS_CRYPT_30300]
[RS_CRYPTO_02201]	The Crypto Stack shall provide interfaces to use symmetric encryption and decryption primitives	[SWS_CRYPT_01501] [SWS_CRYPT_01502] [SWS_CRYPT_01503] [SWS_CRYPT_01504] [SWS_CRYPT_01506] [SWS_CRYPT_01508] [SWS_CRYPT_01651] [SWS_CRYPT_01653] [SWS_CRYPT_01654] [SWS_CRYPT_01655] [SWS_CRYPT_01656] [SWS_CRYPT_01657] [SWS_CRYPT_01658] [SWS_CRYPT_01659] [SWS_CRYPT_01660] [SWS_CRYPT_01661] [SWS_CRYPT_01662] [SWS_CRYPT_02123] [SWS_CRYPT_20742] [SWS_CRYPT_20744] [SWS_CRYPT_23600] [SWS_CRYPT_23601] [SWS_CRYPT_23700] [SWS_CRYPT_23701] [SWS_CRYPT_23716] [SWS_CRYPT_23801] [SWS_CRYPT_23802] [SWS_CRYPT_24001] [SWS_CRYPT_24011] [SWS_CRYPT_24012] [SWS_CRYPT_24013] [SWS_CRYPT_24014] [SWS_CRYPT_40963] [SWS_CRYPT_40964]

▽

△

Requirement	Description	Satisfied by
[RS_CRYPTO_02202]	The Crypto Stack shall provide interfaces to use asymmetric encryption and decryption primitives	[SWS_CRYPT_02700] [SWS_CRYPT_02701] [SWS_CRYPT_02702] [SWS_CRYPT_02703] [SWS_CRYPT_02704] [SWS_CRYPT_02705] [SWS_CRYPT_02726] [SWS_CRYPT_20750] [SWS_CRYPT_20751] [SWS_CRYPT_20754] [SWS_CRYPT_20755] [SWS_CRYPT_20800] [SWS_CRYPT_20801] [SWS_CRYPT_20811] [SWS_CRYPT_20812] [SWS_CRYPT_21000] [SWS_CRYPT_21001] [SWS_CRYPT_21011] [SWS_CRYPT_21012] [SWS_CRYPT_22200] [SWS_CRYPT_22700] [SWS_CRYPT_22701] [SWS_CRYPT_22702] [SWS_CRYPT_22711] [SWS_CRYPT_22712] [SWS_CRYPT_23200] [SWS_CRYPT_23201] [SWS_CRYPT_23215] [SWS_CRYPT_40966]
[RS_CRYPTO_02203]	The Crypto Stack shall provide interfaces to use message authentication code primitives	[SWS_CRYPT_01200] [SWS_CRYPT_01201] [SWS_CRYPT_01202] [SWS_CRYPT_01203] [SWS_CRYPT_01204] [SWS_CRYPT_01207] [SWS_CRYPT_01210] [SWS_CRYPT_01211] [SWS_CRYPT_20746] [SWS_CRYPT_22100] [SWS_CRYPT_22101] [SWS_CRYPT_22115] [SWS_CRYPT_22116]
[RS_CRYPTO_02204]	The Crypto Stack shall provide interfaces to use digital signature primitives	[SWS_CRYPT_00902] [SWS_CRYPT_02400] [SWS_CRYPT_02408] [SWS_CRYPT_02409] [SWS_CRYPT_02410] [SWS_CRYPT_02411] [SWS_CRYPT_02412] [SWS_CRYPT_02413] [SWS_CRYPT_02414] [SWS_CRYPT_02415] [SWS_CRYPT_02416] [SWS_CRYPT_02417] [SWS_CRYPT_02418] [SWS_CRYPT_02419] [SWS_CRYPT_02420] [SWS_CRYPT_02422] [SWS_CRYPT_20003] [SWS_CRYPT_20754] [SWS_CRYPT_20755] [SWS_CRYPT_20756] [SWS_CRYPT_20757] [SWS_CRYPT_22200] [SWS_CRYPT_22201] [SWS_CRYPT_22215] [SWS_CRYPT_23200] [SWS_CRYPT_23201] [SWS_CRYPT_23500] [SWS_CRYPT_23501] [SWS_CRYPT_23511] [SWS_CRYPT_23512] [SWS_CRYPT_23513] [SWS_CRYPT_24100] [SWS_CRYPT_24101] [SWS_CRYPT_24111] [SWS_CRYPT_24112] [SWS_CRYPT_24114] [SWS_CRYPT_41027] [SWS_CRYPT_41039] [SWS_CRYPT_41040] [SWS_CRYPT_41041]
[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_20747] [SWS_CRYPT_21100] [SWS_CRYPT_21101] [SWS_CRYPT_21115] [SWS_CRYPT_21116]
[RS_CRYPTO_02206]	The Crypto Stack shall provide interfaces to configure and use random number generation	[SWS_CRYPT_00500] [SWS_CRYPT_00501] [SWS_CRYPT_00502] [SWS_CRYPT_00503] [SWS_CRYPT_00504] [SWS_CRYPT_00505] [SWS_CRYPT_00506] [SWS_CRYPT_00507] [SWS_CRYPT_00508] [SWS_CRYPT_20741] [SWS_CRYPT_22900] [SWS_CRYPT_22901] [SWS_CRYPT_22911] [SWS_CRYPT_22912] [SWS_CRYPT_22914] [SWS_CRYPT_22915] [SWS_CRYPT_30098] [SWS_CRYPT_40983] [SWS_CRYPT_40988] [SWS_CRYPT_40989] [SWS_CRYPT_40990]

▽

△

Requirement	Description	Satisfied by
[RS_CRYPTO_02207]	The Crypto Stack shall provide interfaces to use authenticated symmetric encryption and decryption primitives	[SWS_CRYPT_01800] [SWS_CRYPT_01801] [SWS_CRYPT_01802] [SWS_CRYPT_01803] [SWS_CRYPT_01804] [SWS_CRYPT_01805] [SWS_CRYPT_01806] [SWS_CRYPT_01807] [SWS_CRYPT_01808] [SWS_CRYPT_01811] [SWS_CRYPT_01820] [SWS_CRYPT_01821] [SWS_CRYPT_01822] [SWS_CRYPT_01823] [SWS_CRYPT_20100] [SWS_CRYPT_20101] [SWS_CRYPT_20316] [SWS_CRYPT_20745] [SWS_CRYPT_41023]
[RS_CRYPTO_02208]	The Crypto Stack shall provide interfaces to use symmetric key wrapping primitives	[SWS_CRYPT_02104] [SWS_CRYPT_02105] [SWS_CRYPT_02106] [SWS_CRYPT_02107] [SWS_CRYPT_02108] [SWS_CRYPT_02109] [SWS_CRYPT_02121] [SWS_CRYPT_02122] [SWS_CRYPT_20743] [SWS_CRYPT_24000] [SWS_CRYPT_40965]
[RS_CRYPTO_02209]	The Crypto Stack shall provide interfaces to use asymmetric key encapsulation primitives	[SWS_CRYPT_03000] [SWS_CRYPT_03002] [SWS_CRYPT_03003] [SWS_CRYPT_03004] [SWS_CRYPT_03005] [SWS_CRYPT_03006] [SWS_CRYPT_03007] [SWS_CRYPT_03008] [SWS_CRYPT_03009] [SWS_CRYPT_20752] [SWS_CRYPT_20753] [SWS_CRYPT_21400] [SWS_CRYPT_21800] [SWS_CRYPT_21801] [SWS_CRYPT_40967] [SWS_CRYPT_40968]
[RS_CRYPTO_02301]	The Crypto Stack API shall provide a standardized header files structure	[SWS_CRYPT_20099] [SWS_CRYPT_30099] [SWS_CRYPT_40099]
[RS_CRYPTO_02302]	The Crypto Stack API shall support a streaming approach	[SWS_CRYPT_10701] [SWS_CRYPT_10710] [SWS_CRYPT_20312] [SWS_CRYPT_20313] [SWS_CRYPT_20314] [SWS_CRYPT_21110] [SWS_CRYPT_21111] [SWS_CRYPT_21112] [SWS_CRYPT_21113] [SWS_CRYPT_21114] [SWS_CRYPT_21115] [SWS_CRYPT_21118] [SWS_CRYPT_22110] [SWS_CRYPT_22111] [SWS_CRYPT_22112] [SWS_CRYPT_22113] [SWS_CRYPT_22114] [SWS_CRYPT_22115] [SWS_CRYPT_23614] [SWS_CRYPT_23615] [SWS_CRYPT_23616] [SWS_CRYPT_23618] [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] [SWS_CRYPT_00006] [SWS_CRYPT_10000] [SWS_CRYPT_20700] [SWS_CRYPT_30100] [SWS_CRYPT_41021] [SWS_CRYPT_41022] [SWS_CRYPT_41029]

▽

△

Requirement	Description	Satisfied by
[RS_CRYPTO_02306]	The Crypto Stack shall support integration with a Public Key Infrastructure (PKI)	[SWS_CRYPT_20001] [SWS_CRYPT_20002] [SWS_CRYPT_20003] [SWS_CRYPT_20004] [SWS_CRYPT_20005] [SWS_CRYPT_20006] [SWS_CRYPT_20007] [SWS_CRYPT_20009] [SWS_CRYPT_20010] [SWS_CRYPT_20011] [SWS_CRYPT_20301] [SWS_CRYPT_20302] [SWS_CRYPT_20303] [SWS_CRYPT_20304] [SWS_CRYPT_20601] [SWS_CRYPT_20602] [SWS_CRYPT_20603] [SWS_CRYPT_20611] [SWS_CRYPT_20612] [SWS_CRYPT_20613] [SWS_CRYPT_20614] [SWS_CRYPT_20615] [SWS_CRYPT_20616] [SWS_CRYPT_20617] [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_40213] [SWS_CRYPT_40214] [SWS_CRYPT_40215] [SWS_CRYPT_40216] [SWS_CRYPT_40217] [SWS_CRYPT_40218] [SWS_CRYPT_40220] [SWS_CRYPT_40300] [SWS_CRYPT_40301] [SWS_CRYPT_40302] [SWS_CRYPT_40311] [SWS_CRYPT_40313] [SWS_CRYPT_40314] [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_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_40626] [SWS_CRYPT_40627] [SWS_CRYPT_40628] [SWS_CRYPT_40629] [SWS_CRYPT_40630] [SWS_CRYPT_40631] [SWS_CRYPT_40632] [SWS_CRYPT_40633] [SWS_CRYPT_40634] [SWS_CRYPT_40635] [SWS_CRYPT_40636] [SWS_CRYPT_40640] [SWS_CRYPT_40641] [SWS_CRYPT_40700] [SWS_CRYPT_40701] [SWS_CRYPT_40702] [SWS_CRYPT_40711] [SWS_CRYPT_40800] ▽

▽

△

Requirement	Description	Satisfied by
		△ [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] [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] [SWS_CRYPT_40992] [SWS_CRYPT_40993] [SWS_CRYPT_40994] [SWS_CRYPT_41028]
[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_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] [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]

▽

△

Requirement	Description	Satisfied by
[RS_CRYPTO_02310]	The Crypto Stack API shall support an efficient mechanism of error states notification	[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] [SWS_CRYPT_41050]
[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] [SWS_CRYPT_00006] [SWS_CRYPT_00009] [SWS_CRYPT_10017] [SWS_CRYPT_20098] [SWS_CRYPT_20099] [SWS_CRYPT_20654] [SWS_CRYPT_20700] [SWS_CRYPT_30001] [SWS_CRYPT_30002] [SWS_CRYPT_30003] [SWS_CRYPT_30099] [SWS_CRYPT_30100] [SWS_CRYPT_30130] [SWS_CRYPT_30403] [SWS_CRYPT_40911] [SWS_CRYPT_41030] [SWS_CRYPT_41033] [SWS_CRYPT_41034] [SWS_CRYPT_41035] [SWS_CRYPT_41054] [SWS_CRYPT_41055]
[RS_CRYPTO_02403]	The Crypto Stack shall support isolating keys and requests	[SWS_CRYPT_22500] [SWS_CRYPT_23800] [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_30400] [SWS_CRYPT_30401] [SWS_CRYPT_30402] [SWS_CRYPT_41058]
[RS_IAM_00010]	Adaptive applications shall only be able to use AUTOSAR Resources when authorized	[SWS_CRYPT_41056]
[RS_Main_00491]	Function Monitoring	[SWS_CRYPT_41042] [SWS_CRYPT_41043] [SWS_CRYPT_41044] [SWS_CRYPT_41045] [SWS_CRYPT_41046] [SWS_CRYPT_41047] [SWS_CRYPT_41048] [SWS_CRYPT_41049] [SWS_CRYPT_41051] [SWS_CRYPT_41052] [SWS_CRYPT_41053] [SWS_CRYPT_41054] [SWS_CRYPT_41055] [SWS_CRYPT_41056] [SWS_CRYPT_41057] [SWS_CRYPT_41058]
[SWS_CORE_10980]	ErrorDomain sub-class accessor function	[SWS_CRYPT_19952]

Table 6.1: Requirements Tracing

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

[SWS_CRYPT_00101]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02110

⌈When `ara::core::Initialize` 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.

⌋

Hint: Access structures may encompass the communication channel between the application process and the stack process or other resource required by the CryptoAPI.

[SWS_CRYPT_41051] Log initialization failed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈Whenever Initialization of crypto structures failed during a call of `ara::core::Initialize`, `FC Crypto` shall log a DltMessage of type InitializationFailed with arguments set to:

- ProcessId: The process failed to initialize crypto

⌋

7.1.2 Shutdown

[SWS_CRYPT_00102]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004, RS_CRYPTO_02007, RS_CRYPTO_02102

⌈When `ara::core::Deinitialize` is called, the `FC Crypto` shall ensure that all open contexts are closed and all occupied ressources are freed.⌋

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

[SWS_CRYPT_00103]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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.⌋

[SWS_CRYPT_41052] Log deInitialization failed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02102, RS_Main_00491

⌈Whenever deInitialization of crypto structures failed during call of `ara::core::Deinitialize`, `FC Crypto` shall log a DltMessage of type DeInitializationFailed with arguments set to:

- ProcessId: The process failed to deInitialize crypto

⌋

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

[SWS_CRYPT_41050] Initialization violation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈The functions `ara::crypto::LoadKeyStorageProvider` and `ara::crypto::LoadX509Provider` shall check that `ara::core::Initialize` has been called successfully and `ara::core::Deinitialize` has not yet been executed. If the check failed, this shall be treated as a violation according to SWS_CORE_00021.⌋

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.

Although only interfaces to the user of `FC Crypto` are standardized in this specification, it is recommended to use standardized interfaces between the HSM and Crypto-Provider implementation. Examples of standardized interfaces are:

- [9, AUTOSAR Classic Crypto Driver]
- [10, PKCS#11]
- [11, Platform Security Architecture (PSA)]

`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.

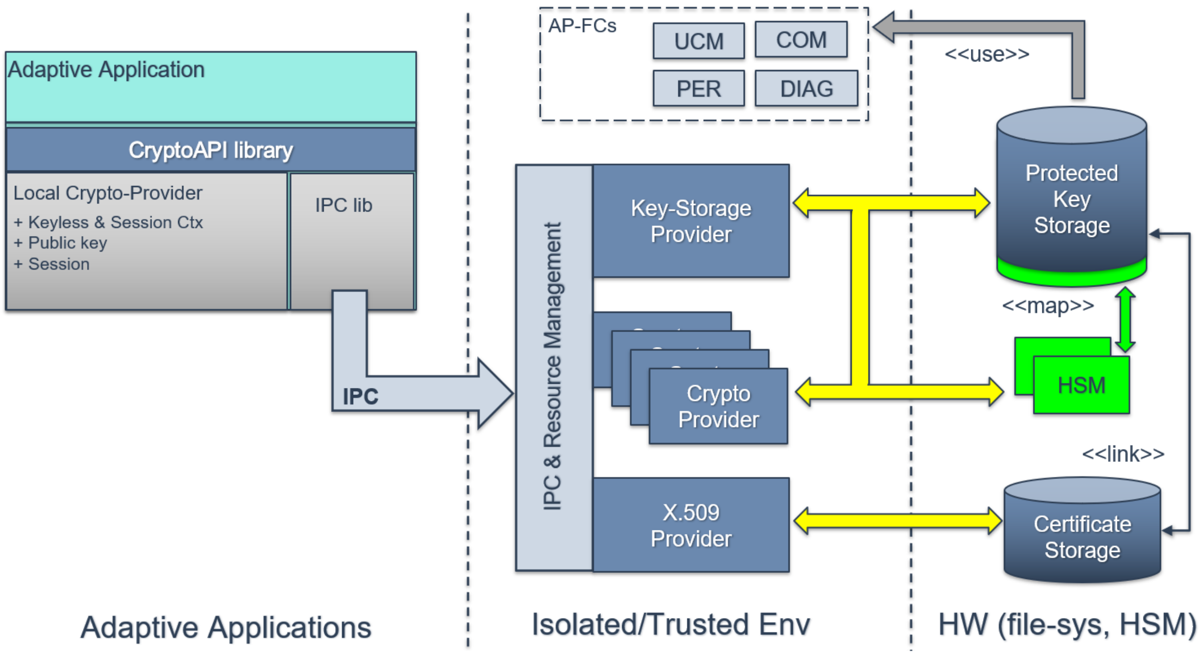

Figure 7.1: High-level CryptoAPI architecture

7.2.1 Integration with Identity and Access Management

To enable access control FC Crypto supports Policy Enforcement Point (PEP) implementation 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 one possible solution is to embed the PEP within the Key Storage Provider and the X.509 Provider. This 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. Another possible solution is to implement the PEP outside of FC Crypto.

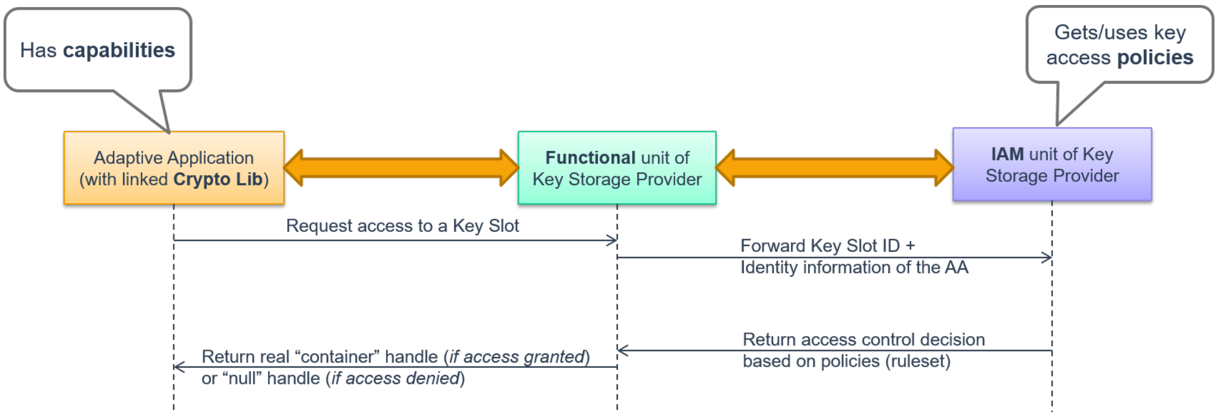
Has capabilities
Gets/uses key access policies
Adaptive Application (with linked Crypto Lib)
Functional unit of Key Storage Provider
IAM unit of Key Storage Provider
Request access to a Key Slot
Forward Key Slot ID + Identity information of the AA
Return access control decision based on policies (ruleset)
Return real "container" handle (if access granted) or "null" handle (if access denied)

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).

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).

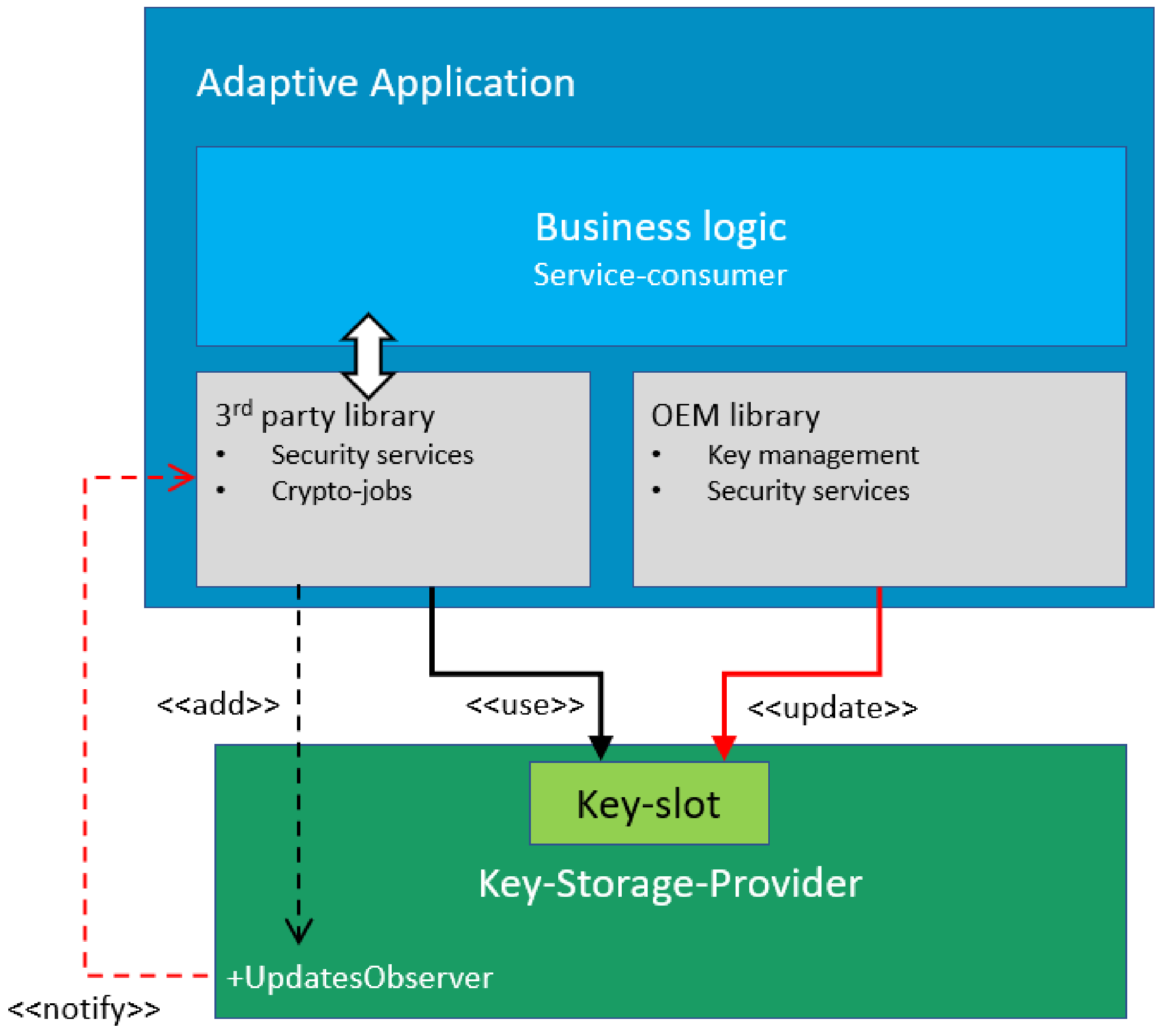

Figure 7.3: Concurrent access to a single Key Slot

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

[SWS_CRYPT_41056] Log Access not granted

Status: DRAFT
Upstream requirements: RS_IAM_00010, RS_Main_00491

⌈Whenever an application attempts to access a crypto resource but it is unauthorized and the IAM return with a "not granted" response, FC Crypto shall log a DltMessage of type ResourceAccessNotGranted with arguments set to

- ProcessId: Process identifier of the process that was not granted access to a crypto resource.
- ResourceInstanceSpecifier: A Provider or KeySlot or any crypto resource instance specifier that the process can not grant access to.

⌋

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.

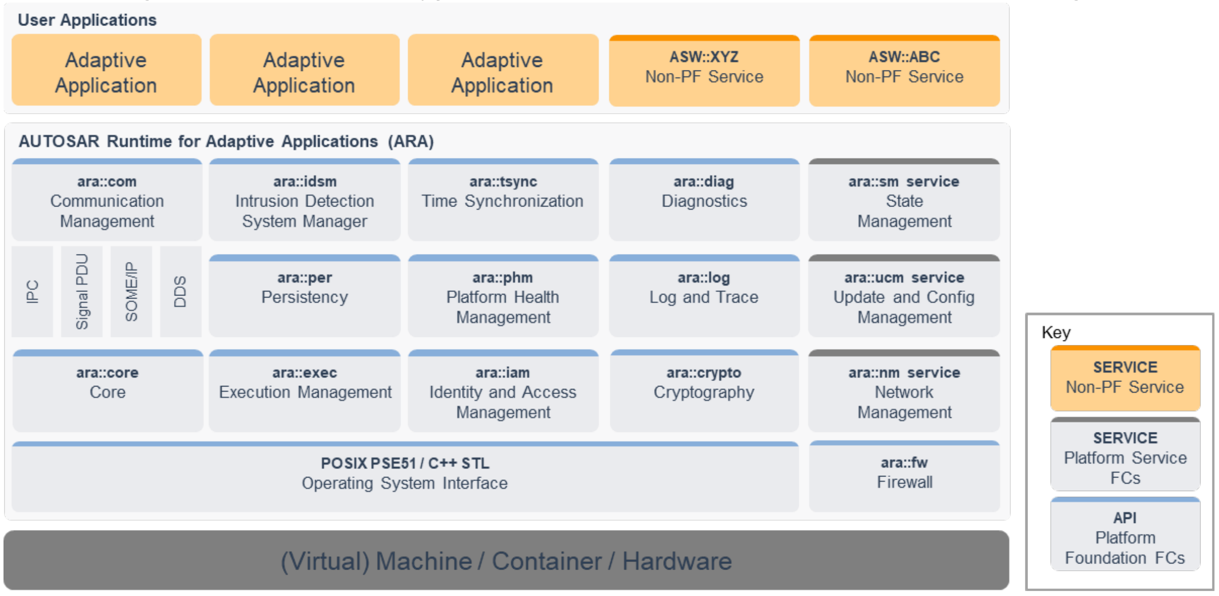
User Applications
Adaptive Application
Adaptive Application
Adaptive Application
ASW::XYZ Non-PF Service
ASW::ABC Non-PF Service
AUTOSAR Runtime for Adaptive Applications (ARA)
ara::com Communication Management
ara::idsm Intrusion Detection System Manager
ara::tsync Time Synchronization
ara::diag Diagnostics
ara::sm service State Management
IPC
Signal PDU
SOME/IP
DDS
ara::per Persistency
ara::phm Platform Health Management
ara::log Log and Trace
ara::ucm service Update and Config Management
ara::core Core
ara::exec Execution Management
ara::iam Identity and Access Management
ara::crypto Cryptography
ara::nm service Network Management
POSIX PSE51 / C++ STL Operating System Interface
ara::fw Firewall
(Virtual) Machine / Container / Hardware
Key
SERVICE Non-PF Service
SERVICE Platform Service FCs
API Platform Foundation FCs

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. The access model for the application key slot will protect the application own resources from being accessed by any 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 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.

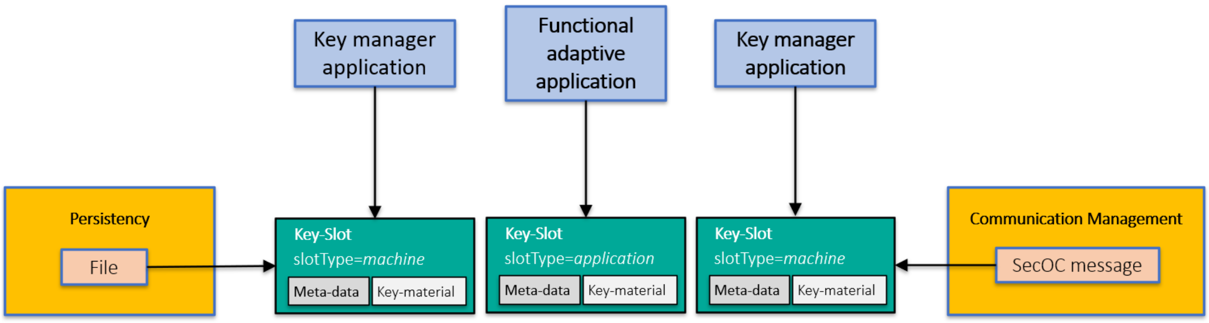

Figure 7.5: Key Slot types and usages

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.

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 Crypto-ProviderInterface. 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.

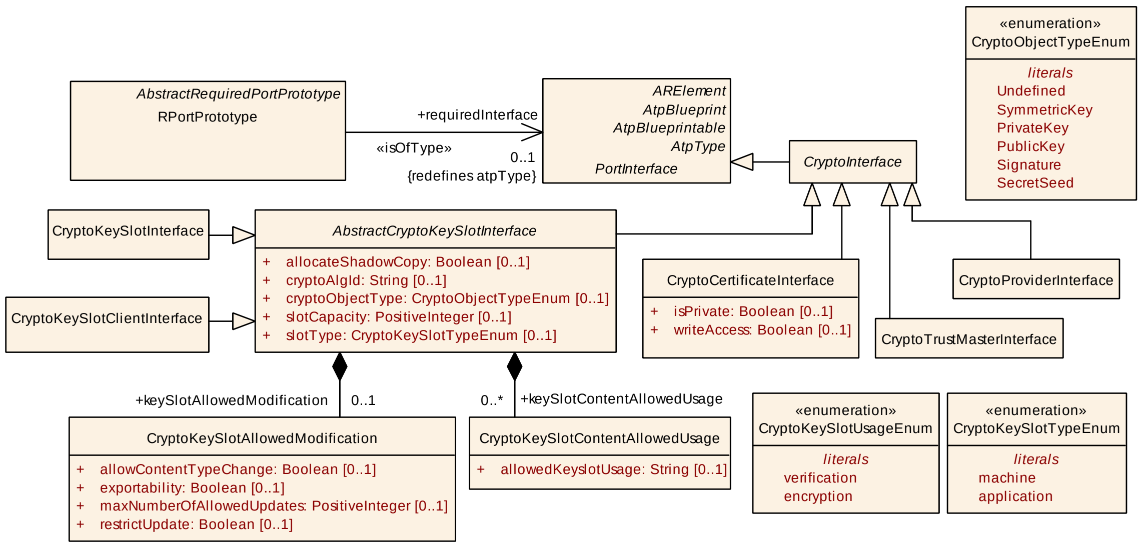
AbstractRequiredPortPrototype
RPortPrototype
+requiredInterface
«isOfType»
0..1
{redefines atpType}
ARElement
AtpBlueprint
AtpBlueprintable
AtpType
PortInterface
CryptoInterface
«enumeration»
CryptoObjectTypeEnum
literals
Undefined
SymmetricKey
PrivateKey
PublicKey
Signature
SecretSeed
CryptoKeySlotInterface
AbstractCryptoKeySlotInterface
+ allocateShadowCopy: Boolean [0..1]
+ cryptoAlgId: String [0..1]
+ cryptoObjectType: CryptoObjectTypeEnum [0..1]
+ slotCapacity: PositiveInteger [0..1]
+ slotType: CryptoKeySlotTypeEnum [0..1]
CryptoKeySlotClientInterface
CryptoCertificateInterface
+ isPrivate: Boolean [0..1]
+ writeAccess: Boolean [0..1]
CryptoProviderInterface
CryptoTrustMasterInterface
+keySlotAllowedModification
0..1
0..*
+keySlotContentAllowedUsage
CryptoKeySlotAllowedModification
+ allowContentTypeChange: Boolean [0..1]
+ exportability: Boolean [0..1]
+ maxNumberOfAllowedUpdates: PositiveInteger [0..1]
+ restrictUpdate: Boolean [0..1]
CryptoKeySlotContentAllowedUsage
+ allowedKeyslotUsage: String [0..1]
«enumeration»
CryptoKeySlotUsageEnum
literals
verification
encryption
«enumeration»
CryptoKeySlotTypeEnum
literals
machine
application

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02305

⌈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.⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02401

⌈The global factory method `ara::crypto::LoadCryptoProvider` shall instantiate a `ara::crypto::cryp::CryptoProvider` identified by the provided `ara::core::InstanceSpecifier`.⌋

[SWS_CRYPT_00006]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02305, RS_CRYPTO_02401

⌈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.⌋

[SWS_CRYPT_00007]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02002

⌈Derived implementations of the interface class `ara::crypto::cryp::CryptoProvider` shall isolate all non-session `Key Material` from the user (`Adaptive Application`).⌋

[SWS_CRYPT_00009]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈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`.⌋

[SWS_CRYPT_10003]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02009

⌈All derived classes of `ara::crypto::cryp::CryptoContext` shall implement the interface `ara::crypto::cryp::CryptoContext::MyProvider`, which shall return a reference to the `ara::crypto::cryp::CryptoProvider` used to create a concrete instance of such a class.⌋

[SWS_CRYPT_41025] Truncation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02009

⌈All derived classes of `ara::crypto::cryp::CryptoContext` that support truncation of a digest returned in an output buffer shall only write to the requested left-most bits of that output buffer. All other bits of a Byte, in case truncation is not on a Byte boundary, as well as all other Bytes of the output buffer shall not be changed.⌋

Some CryptoAPI interfaces that produce digests such as hashes or message authentication codes also support truncation. Truncation refers to the extraction of only a part of the produced digest. The CryptoAPI implements truncation as specified in NIST publication FIPS 180-4 and SP 800-107, i.e. a user chosen number of left-most bits in a bit string representing the digest. A bit string is defined as "An ordered sequence of 0 and 1 bits. In this Recommendation, the leftmost bit is the most significant bit of the string. The rightmost bit is the least significant bit of the string".

[SWS_CRYPT_41022] Get Provider Identification

Status: DRAFT
Upstream requirements: RS_CRYPTO_02305

⌈The interface ara::crypto::cryp::CryptoProvider::GetProviderID shall parse the optionally provided input data of parameter in and write CryptoProvider specific identification data into the provided parameter idData or return

- kUnsupported, if the instantiated CryptoProvider does not support identification.
- kInsufficientCapacity, if the provided ReadWriteMemRegion parameter idData is insufficient in size to hold the identification data.

⌋

[SWS_CRYPT_00500]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈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. The interface shall return

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

⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈If ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx is called to create a global-state ara::crypto::cryp::RandomGeneratorCtx, the requested RandomGeneratorCtx shall be returned without modification of its state.⌋

[SWS_CRYPT_00601]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈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.

⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈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.

⌋

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

[SWS_CRYPT_01200]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈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.

⌋

[SWS_CRYPT_40963]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈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.

⌋

[SWS_CRYPT_40964]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02107, RS_CRYPTO_02201

⌈The interface ara::crypto::cryp::CryptoProvider::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

⌋

[SWS_CRYPT_01806]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_40965]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02208

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_02400]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_02408]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈The interface ara::crypto::cryp::CryptoProvider::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..

⌋

[SWS_CRYPT_02409]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_02410]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈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.

⌋

[SWS_CRYPT_40966]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02202

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_40967]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02209

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_40968]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02209

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_40962]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈The interfaces ara::crypto::cryp::CryptoProvider::GeneratePrivateKey,ara::crypto::cryp::CryptoProvider::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 attribute isSession, e.g. as isSession. 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.

⌋

[SWS_CRYPT_40969]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈The interface ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_40970] Translation of common name to vendor identifier
Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈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 kInvalidArgument, if the provided primitive name is not supported.⌋

[SWS_CRYPT_40971] Translation of identifier to name

Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈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.⌋

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.

[SWS_CRYPT_41054] Log missing necessary configuration

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401, RS_Main_00491

⌈Whenever Startup of FC Crypto failed due to missing or incorrect configuration which leads to the inability of the crypto module to connect to one of the providers. FC Crypto shall log DltMessage of type ConfigurationMissesNecessaryInformation with arguments set to:

- ProviderInstanceSpecifier: Instance specifier of the Provider that misses the necessary configuration to be found.

⌋

[SWS_CRYPT_41055] Log Crypto Provider loading failed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401, RS_Main_00491

⌈Whenever Loading a Crypto Provider failed when calling ara::crypto::LoadCryptoProvider, FC Crypto shall log a DltMessage of type CryptoProviderLoadingFailed with arguments set to:

- CryptoProviderInstanceSpecifier: Crypto Provider instance specifier.

⌋

[SWS_CRYPT_41057] Log Context not supported

Status: DRAFT
Upstream requirements: RS_Main_00491

⌈Whenever an application attempts to create a context but the Crypto Provider is not supporting this context, FC Crypto shall log a DltMessage of type CreateContextUnsupported with arguments set to:

- CryptoProviderInstanceSpecifier: InstanceSpecifier of the Crypto Provider.
- AlgorithmId: The given algorithm id.

⌋

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 [12, BSIDRNG: Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators].

[SWS_CRYPT_00501]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈If a Crypto Provider provides one or more RNG implementations, one RNG implementation shall be documented as the default and this default RNG shall be used with ara::crypto::cryp::CryptoProvider::GenerateRandomData.⌋

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::RandomGeneratorCtx'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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02206

⌈If a ara::crypto::cryp::RandomGeneratorCtx 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 kUnsupported without modifying the global state or local state.⌋

[SWS_CRYPT_00503]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02206

⌈ara::crypto::cryp::RandomGeneratorCtx::Seed, and ara::crypto::cryp::RandomGeneratorCtx::SetKey shall return kUsageViolation without modifying the state, if they are called with a Symmetric Key or a SecretSeed without the allowed usage flag kAllowRngInit.⌋

[SWS_CRYPT_40988]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02206

⌈The interface ara::crypto::cryp::RandomGeneratorCtx::Seed shall apply the provided data as a seed value for random number generation.⌋

[SWS_CRYPT_40989]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02206

⌈If a ara::crypto::cryp::RandomGeneratorCtx instance supports keyed random number generation, the interface ara::crypto::cryp::RandomGeneratorCtx::SetKey shall use the provided key-material for random number generation.⌋

[SWS_CRYPT_40990]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈The interface ara::crypto::cryp::RandomGeneratorCtx::AddEntropy shall use the provided data as additional entropy for random number generation.⌋

How global-state ara::crypto::cryp::RandomGeneratorCtxs 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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈If ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx is called to create a local-state ara::crypto::cryp::RandomGeneratorCtx, the internal state of the created ara::crypto::cryp::RandomGeneratorCtx shall be seeded by FC Crypto before returning.⌋

While this enables applications to create a ready-to-go ara::crypto::cryp::RandomGeneratorCtx, 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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈If ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx is called to create a local-state ara::crypto::cryp::RandomGeneratorCtx but the context currently cannot be seeded, ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx shall return kBusyResource.⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈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.

⌋

[SWS_CRYPT_00508]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈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.⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈The function ara::crypto::cryp::CryptoProvider::GenerateRandomData shall generate random data from the default random data source of the CryptoProvider and fill the provided output buffer (span) with random data. The interface shall return kBusyResource, if the requested number of random Bytes cannot be provided.⌋

7.4.1.2 Key Derivation Function (KDF)

According to [13], [14], [15], and [16] 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] Symmetric encryption based KDF

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈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.⌋

[SWS_CRYPT_00608]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈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.

⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈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.

⌋

[SWS_CRYPT_00610]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈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.⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02101

⌈The interfaces `ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveKey` and `ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveSeed` shall execute the configured key derivation algorithm and return the derived `Key Material` as a `ara::crypto::cryp::SymmetricKey` or `ara::crypto::cryp::SecretSeed` respectively. All allowed usage flags of these derived objects shall be set to false. If source key-material was provided as a RestrictedUseObject, allowed usage flags shall be copied from their corresponding derived allowed usage counter-part. If source key-material is provided as a ReadOnlyMemRegion, the user provided allowed usage shall be set. If further derivation of the already derived key-material is specified by the source key-material (kAllowDerivedKdfMaterial), all derived allowed usage flags of the source RestrictedUseObject shall be copied to the derived RestrictedUseObject. User provided allowed usage may set flags to true for which no derived allowed usage of the source key-material exists, but may not set flags to true for which a corresponding derived allowed usage is set to false.⌋

As a general rule it is foreseen to propagate the derivability of keys, i.e. a source key may specify that derived keys can be derived further. Therefore, all kAllowDerivedxxx like kAllowDerivedVerification of derived key-material are copied from the source to the derived key. In contrast, every derived allowed usage flag of the source RestrictedUseObject (e.g. kAllowDerivedKdfMaterial) is copied to the corresponding flag on the derived RestrictedUseObject (e.g. kAllowKdfMaterial). The user may only further restrict usage of derived key-material, but not expand it, e.g. kAllowDerivedKdfMaterial==FALSE on the source key means kAllowKdfMaterial==FALSE on the derived key, a user may not specify kAllowKdfMaterial==TRUE in this case. Flags that do not have a corresponding derived usage flag of the source key may be specified by the user as TRUE (e.g. kAllowExport); if the user does not specify allowed usage such flags default to FALSE. Note: if the source key-material is a ReadOnlyMemRegion and the user does not provide allowed usage flags, the derived key may not be used in any transformation (all usage is forbidden).

[SWS_CRYPT_40944]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈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::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.
- `kUsageViolation` error, if the allowed usage flags of the provided `ara::crypto::cryp::RestrictedUseObject` are more restrictive than the allowed usage flags previously set by `ara::crypto::cryp::KeyDerivationFunctionCtx::Init`.
- `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

⌋

[SWS_CRYPT_40945]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈The interface `ara::crypto::cryp::KeyDerivationFunctionCtx::Init` shall configure the key derivation by setting the provided `targetKeyId`, `ara::crypto::CryptoAlgId`, and optionally usage flags and context label of the derived key. If no usage flags are provided, kAllowKdfMaterialAnyUsage shall be used instead. 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.
- `kInvalidArgument` error, if the provided `targetAlgId` does not specify a symmetric key algorithm.

⌋

[SWS_CRYPT_40946]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02101

⌈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.

⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈The ara::crypto::cryp::HashFunctionCtx shall implement hashing.

⌋

[SWS_CRYPT_00903]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈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.⌋

[SWS_CRYPT_00908] Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈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.

⌋

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] Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈The functions `ara::crypto::cryp::HashFunctionCtx::Update`, `ara::crypto::cryp::HashFunctionCtx::Update`, `ara::crypto::cryp::HashFunctionCtx::Update` shall implement the configured hash algorithm calculation.⌋

[SWS_CRYPT_00909] Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈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.⌋

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] Finish

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈The function `ara::crypto::cryp::HashFunctionCtx::Finish` shall finalize the hash value calculation and write the hash value into the provided output buffer. I.e. no more data may be provided by `Update`. The interface shall return the number of Bytes written or

- `CryptoErrorDomain::kProcessingNotStarted`, if `Start` has not been successfully called before.
- `CryptoErrorDomain::kInvalidUsageOrder`, if `Update` has not been called successfully after the last call to `Start`.
- `CryptoErrorDomain::kInsufficientCapacity`, if the provided output buffer does not have sufficient capacity to hold the digest

⌋

[SWS_CRYPT_00910]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈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.⌋

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] Retrieving the hash value

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈The function `ara::crypto::cryp::HashFunctionCtx::GetDigest` shall write the finalized hash value or the requested left-most bits of the hash value, if the application requested truncation, into the provided output buffer and return the number of Bytes written.⌋

[SWS_CRYPT_00919] Signalization of missing finalization error

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈The function `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.⌋

7.4.1.4 Message Authentication Code (MAC)

According to the ISO-9797 [17] `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 [18] and in [19]. 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 [18] and in [19].

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02203

⌈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.⌋

[SWS_CRYPT_01201]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02203

⌈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 iv parameter is provided.

⌋

[SWS_CRYPT_01203] Start

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02203

⌈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.

⌋

[SWS_CRYPT_01204] Update
Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈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.

⌋

[SWS_CRYPT_01207] Finish
Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈The function ara::crypto::cryp::MessageAuthnCodeCtx::Finish shall finalize the MAC calculation, After the call of this function no more data can be provided by callingara::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.

⌋

[SWS_CRYPT_01210] GetDigest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈The function ara::crypto::cryp::MessageAuthnCodeCtx::GetDigest shall write the last calculated message authentication code or, if truncation is requested, the left-most requested bits into the provided output buffer and return the number of Bytes written or

- kProcessingNotFinished, if the digest calculation was not finished by a call of the Finish() method.
- kUsageViolation, if the buffered digest belongs to a MAC/HMAC/AE/AEAD context initialized by a key without kAllowSignature permission.
- kInsufficientCapacity, if the provided output buffer does not have sufficient capacity

⌋

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] SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈The function ara::crypto::cryp::MessageAuthnCodeCtx::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.
- kInvalidArgument error, if the provided transformation direction is not allowed in Message Authn Code algorithm context

⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈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 `kIncompatibleObject` error, if the provided `Symmetric Key` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.

- `SetKey` shall return a `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`.

- `SetKey` shall return a `kInvalidArgument` error, if the provided transformation direction is not allowed in Symmetric BlockCipher algorithm context.

⌋

[SWS_CRYPT_01501]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02107, RS_CRYPTO_02201

⌈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.⌋

[SWS_CRYPT_01508]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02107, RS_CRYPTO_02201

⌈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.

⌋

[SWS_CRYPT_01506]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02107, RS_CRYPTO_02201

⌈The interface `ara::crypto::cryp::SymmetricBlockCipherCtx::GetCryptoService` shall return a unique pointer to the `ara::crypto::cryp::CryptoService` associated with this context.⌋

[SWS_CRYPT_01503]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈The interface `ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlock` shall apply the configured transformation (encryption or decryption) to the provided `ara::crypto::ReadOnlyMemRegion`, write the result to the provided output buffer and return the number of Bytes written or

- `kUninitializedContext`, if `ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey` was never called.
- `kInsufficientCapacity`, if the provided output buffer does not have sufficient capacity.
- `kInvalidInputSize`, if the provided input buffer is larger than the block size for kEncrypt.

- kInvalidInputSize, if the context requires padding and the size of the input buffer is not exactly once or twice the block size for kDecrypt.
- kInvalidInputSize, if the context does not support padding and the input buffer size is smaller than the block size.

⌋

[SWS_CRYPT_01504]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈The interface ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlocks shall apply the configured transformation (encryption or decryption) to the provided ara::crypto::ReadOnlyMemRegion, write the result into the provided output buffer and return the number of Bytes written or

- CryptoErrorDomain::kUninitializedContext, if ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey was never called.
- CryptoErrorDomain::kInvalidInputSize, if the size of the input buffer is not a multiple of the block-size.
- CryptoErrorDomain::kInsufficientCapacity, if the provided output buffer does not have sufficient capacity.

⌋

Note: ProcessBlocks shall not apply padding, but instead the size of the input buffer must be a multiple of the block-size.

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02107, RS_CRYPTO_02201

⌈The interface `ara::crypto::cryp::StreamCipherCtx::GetBlockService` shall return a unique pointer to the `ara::crypto::cryp::BlockService` associated with this context.⌋

[SWS_CRYPT_01658]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈The interface `ara::crypto::cryp::StreamCipherCtx::CountBytesInCache` shall return the number of input data bytes currently held in the context cache.⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈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 `kIncompatibleObject` error, if the provided `Symmetric Key` belongs to a different `ara::crypto::cryp::CryptoProvider` instance.
- `SetKey` shall return a `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`.
- `SetKey` shall return a `kInvalidArgument` error, if the provided transformation direction is not allowed in StreamCipher algorithm context.

⌋

[SWS_CRYPT_01660]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02201

⌈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.

⌋

[SWS_CRYPT_01661]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02201

⌈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.⌋

Some operation modes of specific `Stream Ciphers` are seekable, e.g., [20, CTR], [21, Salsa20], or [22, 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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02201

⌈The interface `ara::crypto::cryp::StreamCipherCtx::IsSeekableMode` shall return TRUE, if the algorithm specified during context creation supports seek operations.⌋

[SWS_CRYPT_01653]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02201

⌈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 boolean parameter `fromBegin` 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.

⌋

[SWS_CRYPT_01654]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02201

⌈The interface `ara::crypto::cryp::StreamCipherCtx::Start` and `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`.

⌋

`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 `ara::crypto::ReadOnlyMemRegion` must be empty.

[SWS_CRYPT_01655]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈The interface ara::crypto::cryp::StreamCipherCtx::ProcessBlocks shall apply the configured transformation (encryption or decryption) to the provided data, write the result into the provided output buffer, and return the number of Bytes written or

- CryptoErrorDomain::kIncompatibleArguments, if the sizes of the input and output buffer are not equal.
- CryptoErrorDomain::kInsufficientCapacity, if the provided output buffer does not have sufficient capacity.
- CryptoErrorDomain::kInvalidUsageOrder, if this interface is called after ara::crypto::cryp::StreamCipherCtx::ProcessBytes has been called.
- CryptoErrorDomain::kInvalidInputSize, if the size of the input buffer is not a multiple of the block-size.
- CryptoErrorDomain::kProcessingNotStarted, if processing was not started by successfully calling Start or has already been terminated by successfully calling FinishBytes.

⌋

Note: for ProcessBlocks the size of the input and output buffer must be a multiple of the block-size.

[SWS_CRYPT_01656]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈The interface ara::crypto::cryp::StreamCipherCtx::ProcessBytes shall apply the configured transformation (encryption or decryption) to the provided data. 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. The interface shall write processed data into the provided output buffer and return the number of Bytes written or

- CryptoErrorDomain::kProcessingNotStarted, if processing was not started by successfully calling Start or has already been terminated by successfully calling FinishBytes.
- CryptoErrorDomain::kInsufficientCapacity, if the provided output buffer does not have sufficient capacity.

⌋

Note: for ProcessBytes the size of the input buffer does not need to be a multiple of the block-size. This means smaller chunks of the input data can be provided through this interface and therefore, it is possible that no data is processed (buffer contains less than the block-size) and no data is written to the output buffer (return 0)!

[SWS_CRYPT_01657]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02201

⌈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. 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 from the result. The interface shall write the result into the output buffer provided and return the number of Bytes written or

- `CryptoErrorDomain::kProcessingNotStarted`, if processing was not started by successfully calling `Start` or has already been terminated by successfully calling `FinishBytes`.
- `CryptoErrorDomain::kInsufficientCapacity`, if the provided output buffer does not sufficient capacity.
- `CryptoErrorDomain::kInvalidInputSize`, if the transformation direction is kDecrypt and available data is not a multiple of the block-size.
- `CryptoErrorDomain::kInvalidInputSize`, if the context does not support padding and available data is not a multiple of the block-size.

⌋

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, [23, PKCS5], [24, PKCS5], [25, PKCS7], or [26, 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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02207

⌈The functions ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData, ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData, ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData shall return:

- a kInvalidUsageOrder error, if ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData has already been called.
- a kProcessingNotStarted error, if ara::crypto::cryp::AuthCipherCtx::Start has not been called before.

⌋

[SWS_CRYPT_01801]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02207

⌈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.⌋

[SWS_CRYPT_01802]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02207

⌈Calling UpdateAssociatedData is optional for the user. In this case the MAC shall be calculated over the confidential data only.⌋

[SWS_CRYPT_01803]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02207

⌈The function ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData shall update the calculation of the MAC with the confidential data.

⌋

[SWS_CRYPT_01804]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈If the transformation direction (ara::crypto::cryp::AuthCipherCtx::GetTransformation)is kEncrypt, ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData shall also encrypt the provided Plaintext data and write the Ciphertext into the provided buffer and return the number of Bytes written.

⌋

[SWS_CRYPT_01805]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈If the transformation direction is kDecrypt and the calculated MAC matches the provided expectedTag, ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData shall also decrypt the provided Ciphertext data, write the Plaintext into the provided buffer, and return the number of Bytes written. If the calculated MAC does not match the provided expectedTag, kAuthTagNotValid error shall be returned instead.⌋

[SWS_CRYPT_41023] Process Confidential Data Errors

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈The functions ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData and ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData shall return:

- a kProcessingNotStarted error, if ara::crypto::cryp::AuthCipherCtx::Start has not been called before.
- a kInvalidInputSize error, if the context algorithm requires input data to be on the block size ara::crypto::cryp::BlockService::GetBlockSize, but the provided input is not a multiple of the block size ara::crypto::cryp::BlockService::GetBlockSize and the context was created without specifying the padding or the transformation direction is kDecrypt.

⌋

[SWS_CRYPT_01807]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈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 kAllowDataDecryption is not set and the transformation direction is CryptoTransform::kDecrypt.
- a kInvalidArgument error, if the provided transformation direction is not allowed in authenticated cipher symmetric algorithm context.

⌋

[SWS_CRYPT_01808]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02207

⌈The functions ara::crypto::cryp::AuthCipherCtx::Start and 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 and iv parameter is provided.
- 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.

⌋

[SWS_CRYPT_01811]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02207

⌈The ara::crypto::cryp::AuthCipherCtx::GetDigest function shall write the calculated MAC as raw data into the output buffer provided, only after the ProcessConfidentialData has been successfully executed. If a truncation length parameter was provided, only the requested left-most bits shall be written.⌋

7.4.1.7 Key Wrapping

Key Wrapping (as defined in [27] and [28]) 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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02208

⌈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.⌋

[SWS_CRYPT_02122]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02208

⌈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 kIncompatibleObject error, if the provided Symmetric Key belongs to a different ara::crypto::cryp::CryptoProvider instance.
- SetKey shall return a kUsageViolation error, if the provided transformation direction (CryptoTransform::kWrap or CryptoTransform::kUnwrap) does not match the ara::crypto::AllowedUsageFlags (kAllowKeyExporting or kAllowKeyImporting) of the provided SymmetricKey.
- SetKey shall return a kInvalidArgument error, if the provided transformation direction is not allowed in Symmetric Key wrapper algorithm context.

⌋

[SWS_CRYPT_02123]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02107, RS_CRYPTO_02201

⌈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.⌋

[SWS_CRYPT_02104]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02208

⌈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.⌋

[SWS_CRYPT_02106]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02208

⌈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.⌋

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] wrap

Status: DRAFT
Upstream requirements: RS_CRYPTO_02208

⌈The interface ara::crypto::cryp::SymmetricKeyWrapperCtx::WrapKeyMaterial shall execute the key-wrap operation on the key-material of the provided ara::crypto::RestrictedUseObject, write the wrapped data into the provided output buffer, and return the number of bytes written or

- CryptoErrorDomain::kInvalidInputSize, if the length of the provided ara::crypto::RestrictedUseObject is unsupported by the algorithm specified during context creation.
- CryptoErrorDomain::kUninitializedContext, if ara::crypto::cryp::SymmetricKeyWrapperCtx::SetKey was never called.
- CryptoErrorDomain::kUsageViolation, if the kAllowExport flag is not set in the ara::crypto::AllowedUsageFlags of the provided ara::crypto::RestrictedUseObject.
- CryptoErrorDomain::kUsageViolation, if the kAllowKeyExporting flag of the ara::crypto::AllowedUsageFlags is not set for the SymmetricKey specified in the SetKey call.
- CryptoErrorDomain::kInsufficientCapacity, if the provided output buffer does not have sufficient capacity.

⌋

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] unwrap

Status: DRAFT
Upstream requirements: RS_CRYPTO_02208

⌈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.⌋

[SWS_CRYPT_02108] unwrap

Status: DRAFT
Upstream requirements: RS_CRYPTO_02208

⌈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.⌋

[SWS_CRYPT_02109] error handling during unwrap

Status: DRAFT
Upstream requirements: RS_CRYPTO_02208

⌈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 kAllowKeyImporting flag of the ara::crypto::AllowedUsageFlags is not set for the Symmetric Key specified in the SetKey call.
- return a CryptoErrorDomain::kIncompatibleObject error, If the unwrapped key-material is incompatible "shorter or longer" with the target crypto algorithm specified by parameter AlgId.

⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈The `ara::crypto::cryp::MsgRecoveryPublicCtx` shall implement digital signature verification with message recovery according to [29].⌋

[SWS_CRYPT_02412]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈The `ara::crypto::cryp::SigEncodePrivateCtx` shall implement digital signature generation with message encoding according to [29].⌋

[SWS_CRYPT_02413]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈The `ara::crypto::cryp::SignerPrivateCtx` shall implement digital signature generation.⌋

[SWS_CRYPT_02414]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈The `ara::crypto::cryp::VerifierPublicCtx` shall implement digital signature verification.⌋

[SWS_CRYPT_01820]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02207

⌈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.

⌋

[SWS_CRYPT_01821]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈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.

⌋

[SWS_CRYPT_01822]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈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.

⌋

[SWS_CRYPT_01823]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈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.
- kIncompatibleObjecterror, 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.

⌋

[SWS_CRYPT_02415] Pre-hashed signing

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈The interfaces ara::crypto::cryp::SignerPrivateCtx::SignPreHashed and ara::crypto::cryp::SignerPrivateCtx::SignPreHashed shall execute the signing algorithm configured for this context without hashing. Both interfaces shall write the computed signature encoded according to the specified FormatId into the provided output buffer and return the number of Bytes written or

- 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 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 parameter hashAlgId is incompatible with the configured signature algorithm.

- kInsufficientCapacity, if the provided output buffer does not have sufficient capacity.
- kUnsupportedFormat, if specified format is not supported.

⌋

Note: hashing has already been applied by the user.

[SWS_CRYPT_02416] Signing

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈The interface ara::crypto::cryp::SignerPrivateCtx::Sign shall execute the signing algorithm configured for this context. The interface shall write the computed signature encoded according to the specified FormatId into the provided output buffer and return the number of Bytes written or

- kInsufficientCapacity, if the provided output buffer does not have sufficient capacity.
- kUnsupportedFormat, if specified format is not supported.
- kUninitializedContext, if ara::crypto::cryp::SignerPrivateCtx::SetKey was not called before.
- kInvalidInputSize, if a supplied ara::crypto::ReadOnlyMemRegion parameter's size is incompatible with the configured signature algorithm.

⌋

[SWS_CRYPT_02417] Pre-hashed verification

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204

⌈The functions ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed and ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed shall execute the verification algorithm configured for this context without hashing. The interfaces shall return

- TRUE, if the verification was successful.
- FALSE, if the verification failed.
- kUnsupportedFormat, if the provided parameter signature cannot be parsed according to the specified FormatId.
- 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 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 parameter hashAlgId is incompatible with the configured signature algorithm.

⌋

Note: hashing has already been applied by the user.

[SWS_CRYPT_41040] Optional context data

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈The interfaces ara::crypto::cryp::SignerPrivateCtx::AddContextData and ara::crypto::cryp::VerifierPublicCtx::AddContextData shall set the provided byte sequence as context data or return kUnsupported, if the configured signature algorithm does not support context data.⌋

Note: some DSAs support optionally providing context data that is included in the signature verification.

[SWS_CRYPT_02418] Truncation of hash value

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈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.⌋

[SWS_CRYPT_02419] Verification

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈The interface ara::crypto::cryp::VerifierPublicCtx::Verify shall execute the verification algorithm configured for this context. The interface shall return

- TRUE, if the verification was successful

- FALSE, if the verification failed
- kUnsupportedFormat, if the provided parameter signature cannot be parsed according to the specified FormatId.
- 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.

⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈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 write this signature with encoded message into the provided output buffer and return the number of Bytes written or

- kInsufficientCapacity, if the provided output buffer does not have sufficient capacity
- 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.

⌋

[SWS_CRYPT_40984]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈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.⌋

[SWS_CRYPT_02422]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈The interface `ara::crypto::cryp::MsgRecoveryPublicCtx::DecodeAndVerify` shall decode the message from the provided signature and verify the message according to the algorithm configured for this context. If verification was successful, the interface shall write the message into the output buffer provided and return the number of Bytes written or

- `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.
- `kInsufficientCapacity`, if the provided output buffer does not have sufficient capacity.

⌋

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] Separation of asymmetric transformation directions

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈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.⌋

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] Creation of DecryptorPrivateCtx and EncryptorPublicCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈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.

⌋

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 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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈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.

⌋

[SWS_CRYPT_02703]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈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.⌋

[SWS_CRYPT_02704] Encrypting

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈The interfaces `ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock`, `ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock` shall execute the encryption operation using the deployed public key, write the encrypted cipher-text into the output buffer provided and return the number of Bytes written.⌋

[SWS_CRYPT_02705] Decrypting

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈The interface `ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock` shall execute the decryption operation using the deployed private key, write the decrypted plain-text into the output buffer provided and return the number of Bytes written.⌋

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] Errors of ProcessBlock

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈The function `ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock` shall return

- `kUninitializedContext` error, if `ara::crypto::cryp::EncryptorPublicCtx::SetKey` was not called before.
- `kInvalidInputSize` error, if the context does not support padding and the length of provided input data is less than the block size of the configured algorithm.

⌋

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 [30], [31], and [32]. The KEK can be derived from the encapsulated Key Material or from randomly generated data by application of a KDF.

[SWS_CRYPT_03000] Keying-Data

Status: DRAFT
Upstream requirements: RS_CRYPTO_02209

⌈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.

⌋

[SWS_CRYPT_03002] Encapsulation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02209

⌈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 write the encapsulated keying-data into the provided output buffer and return the number of Bytes written 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.
- kInsufficientCapacity, if the provided output buffer does not have sufficient capacity

⌋

[SWS_CRYPT_03003] Key Decapsulation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02209

⌈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.
- kIncompatibleObject, If the decapsulated keying-data is incompatible "shorter or longer" with the target crypto algorithm specified by parameter keyingDataAlgId.

⌋

[SWS_CRYPT_03004] Seed Decapsulation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02209

⌈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.

⌋

[SWS_CRYPT_03005]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02209

⌈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.

⌋

[SWS_CRYPT_03006]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02209

⌈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.

⌋

[SWS_CRYPT_03007]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02209

⌈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.⌋

[SWS_CRYPT_03008]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02209

⌈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).

⌋

[SWS_CRYPT_03009]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02209

⌈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.⌋

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 missuse. 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 [33] is the common used key agreement mechanism.

[SWS_CRYPT_03311] Encryption algorithm

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02101

⌈The FC Crypto shall provide an encryption algorithm, which matches the chosen public-private key pair and the key exchange schema.⌋

[SWS_CRYPT_03300] Ephemeral key usage

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02101

⌈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.⌋

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.

[SWS_CRYPT_03312] SetKey

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02102

⌈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.

⌋

[SWS_CRYPT_03313]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02103

⌈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.⌋

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] Seed agreement

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104

⌈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.

⌋

[SWS_CRYPT_03302] Key agreement

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105

⌈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.

⌋

[SWS_CRYPT_03303] Key agreement - no KDF

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105

⌈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.⌋

[SWS_CRYPT_03304] Key agreement - optional call parameters

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105

⌈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.⌋

[SWS_CRYPT_03305] Key agreement - KDF

Status: DRAFT
Upstream requirements: RS_CRYPTO_02115

⌈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.⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈The ara::crypto::cryp::CryptoContext::GetCryptoPrimitiveId shall return a ara::crypto::cryp::CryptoPrimitiveId of the current used cryptographic algorithm.⌋

[SWS_CRYPT_03905]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈The `ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveName` shall return the common name of the current used cryptographic algorithm.⌋

[SWS_CRYPT_03906]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈The `ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveId` shall return the `ara::crypto::cryp::CryptoPrimitiveId` of the current used cryptographic algorithm.⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈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.⌋

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] Loading cryptographic material

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105, RS_CRYPTO_02112, RS_CRYPTO_02113

⌈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.

⌋

[SWS_CRYPT_40947]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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.⌋

[SWS_CRYPT_40948]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈The interface ara::crypto::IOInterface::GetCapacity shall return capacity of the underlying resource in bytes.⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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.⌋

[SWS_CRYPT_40950]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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.⌋

[SWS_CRYPT_40951]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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, `kEmptyContainer` shall be returned.⌋

[SWS_CRYPT_40952]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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.

⌋

[SWS_CRYPT_40953]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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.⌋

[SWS_CRYPT_40954]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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.

⌋

[SWS_CRYPT_40955]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈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.⌋

[SWS_CRYPT_40956]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈The interface `ara::crypto::IOInterface::IsVolatile` shall only return TRUE, if this IOInterface points to a VolatileTrustedContainer.⌋

[SWS_CRYPT_40957]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈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.⌋

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] Exporting secure objects

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈The function `ara::crypto::cryp::CryptoProvider::ExportSecuredObject` shall serialize the provided CryptoObject and apply the transformation specified by the provided SymmetricKeyWrapperCtx. The function shall write the serialized data into the provided output buffer and return the number of Bytes written to the output buffer 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
- kInsufficientCapacity if the provided parameter @c out does not have sufficient capacity

⌋

[SWS_CRYPT_04213]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈The function ara::crypto::cryp::CryptoProvider::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 write the serialized data into the provided output buffer and return the number of Bytes written to the output buffer 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.
- kInsufficientCapacity if the provided parameter @c out does not have sufficient capacity

⌋

[SWS_CRYPT_04203] Exporting public objects

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈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 write the serialized data into the output buffer provided and return the number of Bytes written 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.
- `kInsufficientCapacity` if the provided parameter @c out does not have sufficient capacity

⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02006

⌈The function `ara::crypto::cryp::CryptoObject::IsExportable` shall only return TRUE, if `kAllowExport` is set in the allowed usage flags of this CryptoObject.⌋

[SWS_CRYPT_41020] CryptoObject Casting

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02005

⌈The interface `ara::crypto::cryp::CryptoObject::Downcast` shall cast the provided generic CryptoObject to the derived object defined by the template ConcreteObject and return a unique smart pointer to this derived object or `kBadObjectType`, if the actual type of the provided crypto object does not match the specified ConcreteObject.⌋

[SWS_CRYPT_04204] Importing secure objects

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈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 write a byte-vector into the provided output buffer according to the secure protocol specified by the provided SymmetricKeyWrapperCtx or

- `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.
- kInsufficientCapacity if the provided output buffer does not have sufficient capacity to hold the response.
- kUnreservedResource if the IOInterface is not opened writable.

⌋

Note: if the secure wrapping protocol does not specify a return value, an empty byte-vector can be returned. For example this is the case with "AES-KeyWrap". Other protocols, such as "SHE/LOADKEY" require a response to be returned that can be sequentially serialized as a byte-vector. In case of "SHE/LOADKEY" the byte-vector may consist of concatenated messages M4 and M5.

The input parameter serialized (ReadOnlyMemRegion) is used to input the confidential data of the secure protocol used. The format of this input data is currently stack-vendor or project specific. The unwrapping scheme however must be specified by the provided SymmetricKeyWrapperCtx (transportContext), e.g. "AES-KeyWrap" or "SHE/LOADKEY".

In case of secure protocols that specify both format and wrapping scheme, such as "SHE/LOADKEY", the input parameter serialized can be used to input the wrapped data (M1 M2 M3). To support additional banks (SHE+), the bank number can be additionally concatenated at the end of this parameter, e.g. (M1 M2 M3 BANK). An alternative could be to specify the bank as part of the AlgId used to create the provided SymmetricKeyWrapperCtx, e.g. "SHE/LOADKEY/BANK-3".

The supported secure protocol and its usage for key-import is defined by the stack-vendor supplied CryptoProvider(s).

The contents of M4 and M5 are described in document [34, AUTOSAR SecureHardwareExtensions].

[SWS_CRYPT_04205] Importing public objects

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈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.

⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈The function ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_04208]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02004

⌈The function ara::crypto::cryp::CryptoProvider::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.⌋

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

[SWS_CRYPT_40959]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈The function ara::crypto::cryp::CryptoProvider::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.

⌋

[SWS_CRYPT_04209]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈The CryptoAPI shall document all importing or exporting by a logging mechanism. This information can be queried.⌋

[SWS_CRYPT_10305]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02002

⌈The interface ara::crypto::cryp::PrivateKey::GetPublicKey shall return an instance of PublicKey corresponding to this PrivateKey.⌋

[SWS_CRYPT_10306]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈The ara::crypto::CryptoObjectUids of corresponding PrivateKey and PublicKey instances shall be the same.⌋

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.

[SWS_CRYPT_41053] Log volatile trusted container insufficient capacity

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004, RS_Main_00491

⌈Whenever allocating a volatile trusted container fails due to insufficient capacity, FC Crypto shall log a DltMessage of type AllocateVolatileContainerInSufficientCapacity with arguments set to:

- VotaltileContainerSize: The required volatile container size in bytes.
- AvailalbleCapacity: The available memory left for creating volatile trusted containers in bytes.

⌋

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] KeySlot Read/Write Access Control

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004, RS_CRYPTO_02305

⌈FC Crypto shall grant a runtime process read/write access to a key-slot, if a CryptoKeySlotToPortPrototypeMapping exists that links:

- The CryptoKeySlot representing the key-slot resource to be accessed.
- The modelled Process, which was used to start this runtime process.

⌋

[SWS_CRYPT_41029] KeySlot Read Access Control

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004, RS_CRYPTO_02305

⌈FC Crypto shall grant a runtime process read access to a key-slot, if a CryptoKeySlotToClientPortPrototypeMapping exists that references:

- The CryptoKeySlot representing the key-slot resource to be accessed in the role keySlot.
- The modelled Process, which was used to start this runtime process, in the role process.

⌋

Assignment of CryptoKeySlots 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.

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.

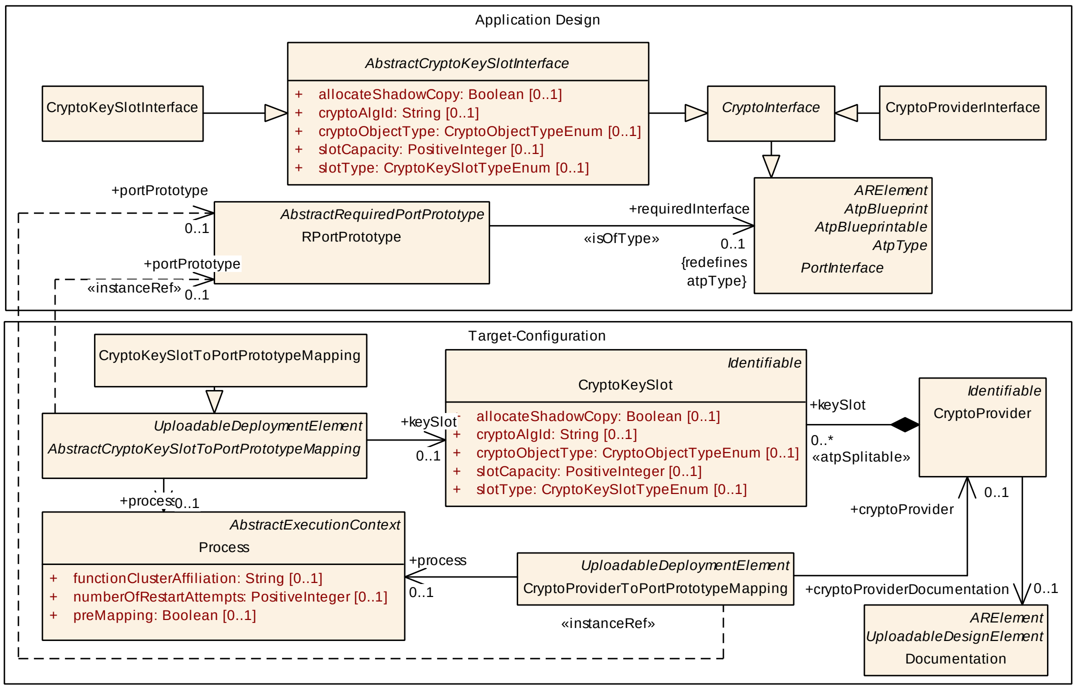
Application Design
AbstractCryptoKeySlotInterface
+ allocateShadowCopy: Boolean [0..1]
+ cryptoAlgId: String [0..1]
+ cryptoObjectType: CryptoObjectTypeEnum [0..1]
+ slotCapacity: PositiveInteger [0..1]
+ slotType: CryptoKeySlotTypeEnum [0..1]
CryptoKeySlotInterface
CryptoInterface
CryptoProviderInterface
+portPrototype
0..1
AbstractRequiredPortPrototype
RPortPrototype
+requiredInterface
«isOfType»
0..1
{redefines atpType}
ARElement
AtpBlueprint
AtpBlueprintable
AtpType
PortInterface
+portPrototype
«instanceRef»
0..1
Target-Configuration
CryptoKeySlotToPortPrototypeMapping
Identifiable
CryptoKeySlot
+ allocateShadowCopy: Boolean [0..1]
+ cryptoAlgId: String [0..1]
+ cryptoObjectType: CryptoObjectTypeEnum [0..1]
+ slotCapacity: PositiveInteger [0..1]
+ slotType: CryptoKeySlotTypeEnum [0..1]
+keySlot
0..*
«atpSplitable»
Identifiable
CryptoProvider
UploadableDeploymentElement
AbstractCryptoKeySlotToPortPrototypeMapping
+keySlot
0..1
+process
0..1
AbstractExecutionContext
Process
+ functionClusterAffiliation: String [0..1]
+ numberOfRestartAttempts: PositiveInteger [0..1]
+ preMapping: Boolean [0..1]
+process
0..1
UploadableDeploymentElement
CryptoProviderToPortPrototypeMapping
+cryptoProvider
0..1
+cryptoProviderDocumentation
0..1
«instanceRef»
ARElement
UploadableDesignElement
Documentation

Figure 7.7: Key deployment

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02008, RS_CRYPTO_02009, RS_CRYPTO_02106

⌈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.⌋

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.

[SWS_CRYPT_41058] Log KeySlot content changed

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02405, RS_Main_00491

⌈Whenever a keyslot content is changed, either added, modified or deleted objects, FC Crypto shall log a DltMessage of type KeySlotContentChanged with arguments set to:

- KeySlotInstanceSpecifier: The key slot instance specifier.

⌋

7.4.2.1 Serializable interface

[SWS_CRYPT_10200]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈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.⌋

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 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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105

⌈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.⌋

[SWS_CRYPT_10401]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02007

⌈Key agreement private context shall provide functionality to produce a common secret seed ara::crypto::cryp::SecretSeed.

⌋

[SWS_CRYPT_10402]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02103

⌈Key agreement private context shall provide functionality to produce a common symmetric key.⌋

7.4.2.3 KeySlots Updates Notification

[SWS_CRYPT_41033] Register UpdatesObserver

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02401

⌈The interface ara::crypto::keys::KeyStorageProvider::RegisterObserver shall take ownership of the ara::crypto::keys::UpdatesObserver instance provided by the application process and keep it in scope until the same process registers a new ara::crypto::keys::UpdatesObserver instance, calls ara::crypto::keys::KeyStorageProvider::UnregisterObserver or FC Crypto is restarted.⌋

[SWS_CRYPT_41034] Single instance of UpdatesObserver

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02401

⌈The interface ara::crypto::keys::KeyStorageProvider::RegisterObserver shall delete a previously registered ara::crypto::keys::UpdatesObserver instance, if the same application process had registered it before.⌋

[SWS_CRYPT_41035] Unregister UpdatesObserver

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈The interface ara::crypto::keys::KeyStorageProvider::UnregisterObserver shall delete the ara::crypto::keys::UpdatesObserver instance previously provided by the application process or silently return, if no ara::crypto::keys::UpdatesObserver instance for this process is available.⌋

It is not required that ara::crypto::keys::KeyStorageProvider::UnregisterObserver returns an error. If an observer of this application process exists, it will be deleted; if not, there is nothing to do. Logging could be used to differentiate between those 2 states.

Implementing synchronization mechanisms can prevent race conditions between ara::crypto::keys::KeyStorageProvider::RegisterObserver and ara::crypto::keys::KeyStorageProvider::UnregisterObserver methods in case the application process calls these interfaces from different threads.

[SWS_CRYPT_41036] Notify KeySlot changes

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈If an ara::crypto::keys::UpdatesObserver instance is available for an application process, the FC Crypto shall call the interface ara::crypto::keys::UpdatesObserver::OnUpdate with a list of unique InstanceSpecifiers representing modified KeySlots to which the same process has subscribed to receive update notifications.⌋

7.4.2.4 KeySlot

[SWS_CRYPT_41037] Cancel updates notification

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈The interface ara::crypto::keys::KeySlot::UnsubscribeFromUpdates shall remove this KeySlot from the list of KeySlots reported to the calling application process on change, i.e. changes of this KeySlot shall not be reported to the calling process.⌋

[SWS_CRYPT_41038] Set updates notification

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈The interface ara::crypto::keys::KeySlot::SubscribeForUpdates shall add this KeySlot to the list of KeySlots reported to the calling application process on change, i.e. changes of this KeySlot shall be reported to the calling process.⌋

Interfaces ara::crypto::keys::KeySlot::SubscribeForUpdates and ara::crypto::keys::KeySlot::UnsubscribeFromUpdates allow the application to respectively select, unselect a KeySlot to be reported on change via the ara::crypto::keys::UpdatesObserver::OnUpdate interface. This does not imply the existence of an ara::crypto::keys::UpdatesObserver instance for the calling application process! The user application may choose the order of deploying an ara::crypto::keys::UpdatesObserver instance and selecting/unselecting KeySlots for change notification. Of course changes are only reported once an ara::crypto::keys::UpdatesObserver instance is made available to FC Crypto by the application.

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 [35] and [36].

[SWS_CRYPT_20000]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02307

⌈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.⌋

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 [37]. 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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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).⌋

[SWS_CRYPT_20002]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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 preconfigured certificates, which are stored along side the configuration, e.g. in ARXML.
- Project specific

⌋

[SWS_CRYPT_20003]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02204, RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20004]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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 [37] and [38, 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.⌋

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] Freedom from interference during update

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02112, RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20006]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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.⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈The X.509 Provider shall store issued certificates in a persistent manner.⌋

[SWS_CRYPT_20009]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20010]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈`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`.⌋

[SWS_CRYPT_20011]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈`X.509 Provider` shall support the Proof-Of-Possession (POP) of the private key.⌋

[SWS_CRYPT_40943]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.

⌋

7.4.3.1 Certificate Signing Request

[SWS_CRYPT_20301]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The X.509 Provider produces the Certificate Signing Request by ara::crypto::x509::X509Provider::CreateCertSignRequest. 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.⌋

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]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20303]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈All meta-information shall be encoded according to the X.509 specification (as given in [37], [39], [40], [41], [42], [43], and [2]).⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

7.4.3.2 Using Certificates

[SWS_CRYPT_41028] Certificate Access Control

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004, RS_CRYPTO_02306

⌈FC Crypto shall grant a runtime process write access to a certificate, if a CryptoCertificateToPortPrototypeMapping exists that links:

- The CryptoCertificate representing the Certificate to be accessed.
- The modelled Process, which was used to start this runtime process.

with CryptoCertificateToPortPrototypeMapping.writeAccess set to true.⌋

[SWS_CRYPT_20601] Importing / Installation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The X.509 Provider provides a mechanism for applications or Functional Clusters to import ara::crypto::x509::X509Provider::Import or install certificates, parts of certification paths, or full certification paths.⌋

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 certification paths with the corresponding PKI. Additionally, the X.509 Provider checks if all Distinguished names and X.509 Extensions are matching the preconfigured 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] Exporting

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The X.509 Provider exports a certificate, a bundle of certificates, a part of a certification path, or a full certification path. The private key of the corresponding export is not included in the export.⌋

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] Getting or Querying

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40912] Querying with wildcards

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40913] Sets of certificates

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

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

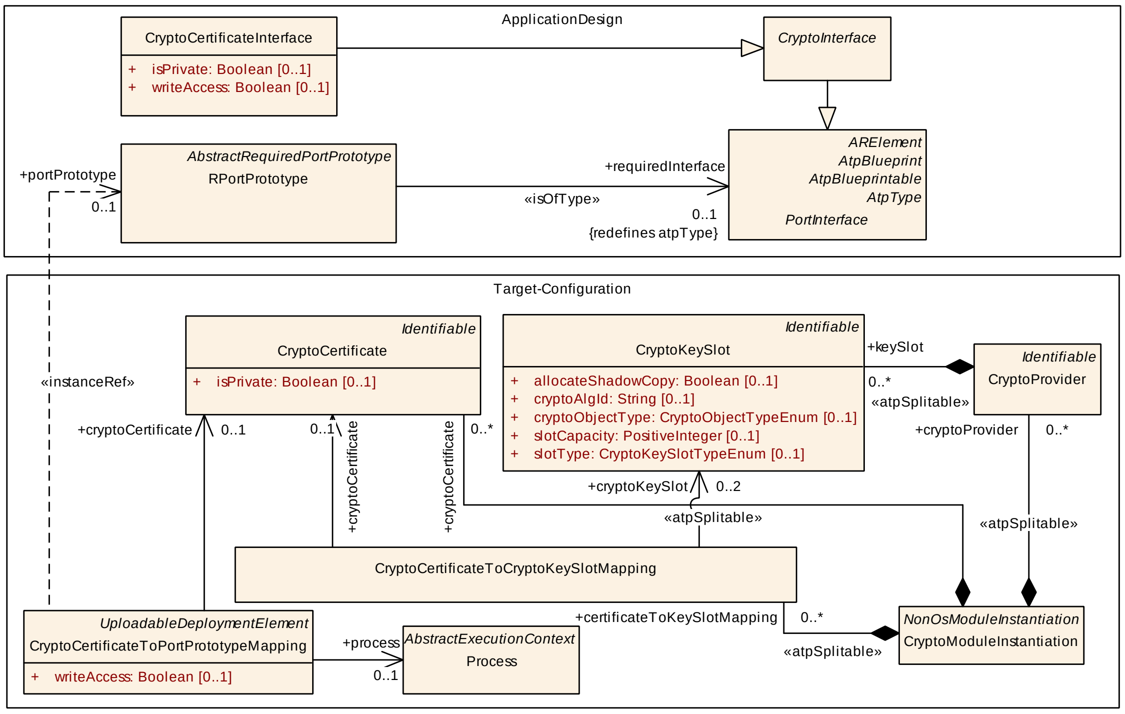

Figure 7.8: Certificate deployment

[SWS_CRYPT_20611] Validation of certification path

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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`.⌋

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]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈The `FC Crypto` allows to encode and decode `ASN.1`-based standard formats (like [44, PKCS#8], [45, PKCS#12]), as specified in [46, X.680], [47, X.682], and [48, X.683]. The `CryptoAPI` allows an application or `Functional Cluster` to select the encoding.⌋

[SWS_CRYPT_20614]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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.⌋

The mapping is provided by `CryptoCertificateToCryptoKeySlotMapping` as shown in 7.8.

[SWS_CRYPT_20615]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20616]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20617]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20618]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20619]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

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] CRL and CTL usage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The X.509 Provider shall support CRL and CTL. The format of CRL and CTL are defined in [38, RFC 5280], [49, RFC 6518], [50, RFC 8398], and [51, 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.⌋

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]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈Given in [38] the CRL can contain two different states:

1. Revoked: certificates that are irreversibly revoked.
2. Hold: certificates that are marked as temporally invalid.

⌋

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] Import

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20904]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The X.509 Provider shall support the online mode to get and update CRL.⌋

[SWS_CRYPT_20905] Verify

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40994]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The function ara::crypto::x509::X509Provider::VerifyCert shall check validity of the provided certificate against the provided root certificate according to [38, RFC 5280] including CRL verification. The function shall stop further processing and return

- kExpired, if the end validity of the provided certificate lies in the past.
- kFuture, if the start validity of the provided certificate lies in the future.

- kInvalid, if the signature of the provided certificate cannot be verified by the provided root certificate.
- kNotAvailable, if the provided root certificate is not referenced from the provided certificate to verify.

⌋

[SWS_CRYPT_40992]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈CRL If the provided certificate is found on one of the certificate revocation lists available to FC Crypto, ara::crypto::x509::X509Provider::VerifyCert shall stop further processing and return kRevoked.⌋

[SWS_CRYPT_40993]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈Certificate chain verification The function ara::crypto::x509::X509Provider::VerifyCertChain shall check validity of the provided certificate chain according to [38, RFC 5280] including CRL verification. The function shall stop further processing and return

- kExpired, if the end validity of any of the provided certificates lies in the past.
- kFuture, if the start validity of any of the provided certificates lies in the future.
- kInvalid, if the signature of any of the provided certificates cannot be verified by the referenced intermediate/root certificate.
- kNotAvailable, if the certificate chain is broken.

⌋

[SWS_CRYPT_20906]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The X.509 Provider shall support the standard protocol, ara::crypto::x509::OcspResponse (as defined in [36, RFC 6960]) and OCSP Stapling (as defined in [52, RFC 6066], [53, RFC 6961], and [54, RFC 8446]), to check if a certificate is revoked. OCSP is an alternative to CRLs.⌋

[SWS_CRYPT_20907]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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 [36, RFC 6960]. The method can be used by an application or Functional Cluster.⌋

[SWS_CRYPT_20908]

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The X.509 Provider shall support request generation for the revocation of certificates.⌋

[SWS_CRYPT_20909] Signalization of revoked certificate by application or functional cluster

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_20910] Internal signalization of revoked certificate

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40972] Configuration Options

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40973]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40974]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈The FC Crypto shall handle the additional behavior to send or request OCSP tickets via the configured mechanism.⌋

[SWS_CRYPT_40975]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈The FC Crypto shall inform the application via an return value that the CRL was updated via configured mechanism.⌋

[SWS_CRYPT_40976]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈The FC Crypto shall inform the application via an return value, that the online information was sent to the configured mechanism.⌋

[SWS_CRYPT_40977]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40978]

Status:	DRAFT
Upstream requirements:	RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40979]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈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.⌋

[SWS_CRYPT_40980]
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈The FC Crypto shall point to the successor of an invalidated certificate, if the OCSP ticket provides a successor.⌋

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] Configuration format for cryptographic algorithms
Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈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 and/or 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.

⌋

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.
- `"CTR/AES-256"` means a CTR-mode stream cipher based on AES algorithm with key-length 256 bit.
- `"CBC/AES-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.
- `"HMAC/SHA-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, I/O-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., `"SIG/RSA-2048,SHA-160/PKCS1.v1_5,EMSA"`)
- `"VER"` – verification primitive (e.g., `"VER/RSA-2048,SHA-160/PKCS1.v1_5,EMSA"`)
- `"ENC"` – encryption primitive (e.g., `"ENC/RSA-2048,MGF1,SHA-160/PKCS1.v1_5,EME"`, `"ENC/RSA-4096,MGF1,SHA2-256/OAEP,EME"`)
- `"DEC"` – decryption primitive (e.g., `"DEC/RSA-2048,MGF1,SHA-160/PKCS1.v1_5,EME"`, `"DEC/RSA-4096,MGF1,SHA2-256/OAEP,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.

7.6 Reporting

7.6.1 Security Events

This functional cluster does not define any security events.

7.6.2 Log Messages

This section lists all non-verbose log messages (i.e., modelled DLT messages) defined by this functional cluster.

[SWS_CRYPT_41042] LogMessage ConfigurationMissesNecessaryInformation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	ConfigurationMissesNecessaryInformation		
Description	The Crypto FC can't connect to none of the configured Providers (CP, KP) or it can't find any of the necessary configuration to correctly function.		
MessageId	0x80002000		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
Provider	A string that identifies the failed provider to load. This can be either "CryptoProvider" or "KeyStorage Provider".	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CRYPT_41043] LogMessage InitializationFailed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	InitializationFailed		
Description	When an application calls ara::core::Initialize, while the Crypto FC can't correctly initialize the structure for this application.		
MessageId	0x80002001		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
ApplicationId	A unique identifier for the calling application, is used to distinguish between different applications.	uint32	NoUnit

⌋

[SWS_CRYPT_41044] LogMessage DeinitializationFailed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	DeinitializationFailed		
Description	When an application calls ara::core::DeInitialize, while the Crypto FC can't correctly DeInitialize the structure for this application.		
MessageId	0x80002002		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
ApplicationId	A unique identifier for the calling application, is used to distinguish between different applications.	uint32	NoUnit

⌋

[SWS_CRYPT_41045] LogMessage CryptoProviderLoadingFailed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	CryptoProviderLoadingFailed		
Description	When a CryptoProvider can't be loaded.		
MessageId	0x80002003		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
CryptoProvider InstanceSpecifier	An instance specifier for the CryptoProvider that failed to load.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CRYPT_41046] LogMessage AllocateVolatileContainerInSufficientCapacity

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	AllocateVolatileContainerInSufficientCapacity		
Description	When an application requires to allocate VTC, but Crypto failed to allocate secure memory for it due to insufficient capacity/memory (e.g. equivilant to failed to malloc).		
MessageId	0x80002004		
MessageType Info	DLT_LOG_WARN		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***

▽

△

Votaltile ContainerSize	The size, in bytes, of the volatile container that failed to allocate due to insufficient memory.	uint32	NoUnit
Available Capacity	The amount of available memory, in bytes, that could be allocated at the time of the allocation rejection.	uint32	NoUnit

⌋

[SWS_CRYPT_41047] LogMessage ResourceAccessNotGranted

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	ResourceAccessNotGranted		
Description	When an application attempts to reach a specific resource but IAM don't grant the access because the application doesn't have an access to the resource.		
MessageId	0x80002005		
MessageType Info	DLT_LOG_WARN		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
ApplicationId	A unique identifier for the calling application, is used to distinguish between different applications.	uint32	NoUnit
Resource InstanceSpecifier	The identifier for the resource that IAM denied access to. This could refer to either a provider or a key slot instance specifier.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CRYPT_41048] LogMessage CreateContextUnsupported

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	CreateContextUnsupported		
Description	When an application attempts to create a context with a given AlgID, but the Context is not supported by this provider at all.		
MessageId	0x80002006		
MessageType Info	DLT_LOG_WARN		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
CryptoProvider InstanceSpecifier	An instance specifier for the CryptoProvider that failed to load.	uint8 [encoding UTF-8]	NoUnit
AlgorithmId	The identifier for the algorithm that was rejected when attempting to create the context, indicates which algorithm is not supported.	uint32	NoUnit

⌋

[SWS_CRYPT_41049] LogMessage KeySlotContentChanged

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110, RS_Main_00491

⌈

Dlt-Message	KeySlotContentChanged		
Description	When, by any means, a KeySlot content is changed, either added, deleted, or modified. Crypto SHALL log the event as info.		
MessageId	0x80002007		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
KeySlotInstance Specifier	An instance specifier for the KeySlot whose contents have been changed indicates which KeySlot is affected.	uint8 [encoding UTF-8]	NoUnit

⌋

7.6.3 Violation Messages

This section lists all violation messages (i.e., DLT messages logged for Violations according to [SWS_CORE_00021]) defined by this functional cluster.

Dlt-Message	ProcessMappingViolation		
Description	Matching InstanceRef exists, but no matching (modelled) Process found that matches the (runtime) process. String format: "Violation detected in {processIdentifier} at {location}: Invalid InstanceSpecifer {instanceSpecifier} in a constructor of class: {className}"		
MessageId	0x80001ffa		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

7.6.4 Production Errors

This functional cluster does not define any production errors (i.e., Diagnostic Events).

8 API specification

This chapter provides a reference of the APIs defined by this functional cluster. The API is described in the following chapters in tables. Table 8.1 explains the content that is described in such an API table.

Kind:	Defines the kind of the declaration that this API table describes. The following values are supported: • class (Declaration of a class) • function (Declaration of a member or non-member function) • struct (Declaration of a structure) • type alias (Declaration of a type alias) • enumeration (Declaration of an enumeration) • variable (Declaration of a variable)	
Header File:	Defines the header file to be included according to [SWS_CORE_90001]	
Forwarding Header File:	Defines the forwarding header file to be included according to [SWS_CORE_90001]	
Scope:	Defines the scope that may be a namespace (in case of a class or non-member function) or a class declaration (in case of a member)	
Symbol:	Entity name	
Thread Safety:	Defines whether a function is thread-safe, not thread-safe, or conditional according to [SWS_CORE_13200] and [SWS_CORE_13202]	
Syntax:	Description of C++ syntax	
Template Param:	Template parameter (0..*)	Template parameter(s) used to parametrize the template
Parameters (in):	Parameter declaration (0..*)	Parameter(s) that are passed to the function
Parameters (out):	Parameter declaration (0..*)	Parameter(s) that are returned to the caller
Return Value:	Return type	Type of the value that the function returns
Exception Safety:	Defines whether a function is exception-safe, not exception safe or conditionally exception safe	
Exceptions:	List of exceptions that may be thrown from the function	
Violations:	List of violations that may occur in the function	
Errors:	Error type (0..*)	List of defined error codes that may be returned by the function with their recoverability class defined in [RS_AP_00160]. APIs can be extended with vendor-specific error codes. These are not part of the AUTOSAR SWS specifications
Description:	Brief description of the function	

Table 8.1: Explanation of an API table

8.1 C++ language binding Crypto Provider

[SWS_CRYPT_20100] Definition of API class ara::crypto::cryp::AuthCipherCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	AuthCipherCtx
Base class:	CryptoContext
Syntax:	`class AuthCipherCtx : public CryptoContext {...};`
Description:	Generalized Authenticated Cipher Context interface. Methods of the derived interface `BufferedDigest` 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()`, `ProcessBytes()` (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!

⌋

[SWS_CRYPT_29030] Definition of API class ara::crypto::cryp::BlockService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

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

⌋

[SWS_CRYPT_20400] Definition of API class ara::crypto::cryp::CryptoContext
Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

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

⌋

[SWS_CRYPT_20500] Definition of API class ara::crypto::cryp::CryptoObject
Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	CryptoObject
Syntax:	`class CryptoObject {...};`
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.

⌋

[SWS_CRYPT_20600] Definition of API class ara::crypto::cryp::CryptoPrimitive Id
Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	CryptoPrimitiveId
Syntax:	`class CryptoPrimitiveId {...};`

▽

△

Description:	Common interface for identification of all Crypto Primitives and their keys & parameters.

⌋

[SWS_CRYPT_20700] Definition of API class ara::crypto::cryp::CryptoProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02305, RS_CRYPTO_02307, RS_CRYPTO_02401

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	CryptoProvider
Syntax:	`class CryptoProvider {...};`
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.

⌋

[SWS_CRYPT_29020] Definition of API class ara::crypto::cryp::CryptoService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

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

⌋

[SWS_CRYPT_20800] Definition of API class ara::crypto::cryp::DecryptorPrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

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

⌋

[SWS_CRYPT_29010] Definition of API class ara::crypto::cryp::DigestService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

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

⌋

[SWS_CRYPT_21000] Definition of API class ara::crypto::cryp::EncryptorPublicCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	EncryptorPublicCtx

▽

△

Base class:	CryptoContext
Syntax:	`class EncryptorPublicCtx : public CryptoContext {...};`
Description:	Asymmetric Encryption Public key Context interface.

⌋

[SWS_CRYPT_29040] Definition of API class ara::crypto::cryp::ExtensionService
Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

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

⌋

[SWS_CRYPT_21100] Definition of API class ara::crypto::cryp::HashFunctionCtx
Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈

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

⌋

[SWS_CRYPT_21300] Definition of API class ara::crypto::cryp::KeyAgreementPrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104

⌈

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

⌋

[SWS_CRYPT_21400] Definition of API class ara::crypto::cryp::KeyDecapsulatorPrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104, RS_CRYPTO_02209

⌈

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

⌋

[SWS_CRYPT_21500] Definition of API class ara::crypto::cryp::KeyDerivationFunctionCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02103

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`

▽

△

Symbol:	KeyDerivationFunctionCtx
Base class:	CryptoContext
Syntax:	`class KeyDerivationFunctionCtx : public CryptoContext {...};`
Description:	Key Derivation Function interface.

⌋

[SWS_CRYPT_21800] Definition of API class ara::crypto::cryp::KeyEncapsulatorPublicCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104, RS_CRYPTO_02209

⌈

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

⌋

[SWS_CRYPT_22100] Definition of API class ara::crypto::cryp::MessageAuthnCodeCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈

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

⌋

[SWS_CRYPT_22200] Definition of API class ara::crypto::cryp::MsgRecoveryPublicCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202, RS_CRYPTO_02204

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	MsgRecoveryPublicCtx
Base class:	CryptoContext
Syntax:	`class MsgRecoveryPublicCtx : public CryptoContext {...};`
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.

⌋

[SWS_CRYPT_22500] Definition of API class ara::crypto::cryp::PrivateKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02002, RS_CRYPTO_02403

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/cryobj/private_key.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	PrivateKey
Base class:	RestrictedUseObject
Syntax:	`class PrivateKey : public RestrictedUseObject {...};`
Description:	Generalized Asymmetric Private Key interface.

⌋

[SWS_CRYPT_22700] Definition of API class ara::crypto::cryp::PublicKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"

▽

△

Scope:	`namespace ara::crypto::cryp`
Symbol:	PublicKey
Base class:	RestrictedUseObject
Syntax:	`class PublicKey : public RestrictedUseObject {...};`
Description:	General Asymmetric Public Key interface.

⌋

[SWS_CRYPT_22900] Definition of API class ara::crypto::cryp::RandomGeneratorCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈

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

⌋

[SWS_CRYPT_24800] Definition of API class ara::crypto::cryp::RestrictedUseObject

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/cryobj/restricted_use_object.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	RestrictedUseObject
Base class:	CryptoObject
Syntax:	`class RestrictedUseObject : public CryptoObject {...};`
Description:	A common interface for all objects supporting the usage restriction.

⌋

[SWS_CRYPT_23000] Definition of API class ara::crypto::cryp::SecretSeed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	SecretSeed
Base class:	RestrictedUseObject
Syntax:	`class SecretSeed : public RestrictedUseObject {...};`
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.

⌋

[SWS_CRYPT_23200] Definition of API class ara::crypto::cryp::SigEncodePrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202, RS_CRYPTO_02204

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	SigEncodePrivateCtx
Base class:	CryptoContext
Syntax:	`class SigEncodePrivateCtx : public CryptoContext {...};`
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.

⌋

[SWS_CRYPT_29000] Definition of API class ara::crypto::cryp::SignatureService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/signature_service.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	SignatureService
Base class:	ExtensionService
Syntax:	`class SignatureService : public ExtensionService {...};`
Description:	Extension meta-information service for signature contexts.

⌋

[SWS_CRYPT_23500] Definition of API class ara::crypto::cryp::SignerPrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	SignerPrivateCtx
Base class:	CryptoContext
Syntax:	`class SignerPrivateCtx : public CryptoContext {...};`
Description:	Signature Private key Context interface.

⌋

[SWS_CRYPT_23600] Definition of API class ara::crypto::cryp::StreamCipherCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	StreamCipherCtx
Base class:	CryptoContext
Syntax:	`class StreamCipherCtx : public CryptoContext {...};`

▽

△

Description:	Generalized Stream Cipher Context interface (it covers all modes of operation).

⌋

[SWS_CRYPT_23700] Definition of API class ara::crypto::cryp::SymmetricBlockCipherCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	SymmetricBlockCipherCtx
Base class:	CryptoContext
Syntax:	`class SymmetricBlockCipherCtx : public CryptoContext {...};`
Description:	Interface of a Symmetric Block Cipher Context with padding.

⌋

[SWS_CRYPT_23800] Definition of API class ara::crypto::cryp::SymmetricKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02403

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/cryobj/symmetric_key.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	SymmetricKey
Base class:	RestrictedUseObject
Syntax:	`class SymmetricKey : public RestrictedUseObject {...};`
Description:	Symmetric Key interface.

⌋

[SWS_CRYPT_24000] Definition of API class ara::crypto::cryp::SymmetricKeyWrapperCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02104, RS_CRYPTO_02208

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	SymmetricKeyWrapperCtx
Base class:	CryptoContext
Syntax:	`class SymmetricKeyWrapperCtx : public CryptoContext {...};`
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.

⌋

[SWS_CRYPT_24100] Definition of API class ara::crypto::cryp::VerifierPublicCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	class
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::cryp`
Symbol:	VerifierPublicCtx
Base class:	CryptoContext
Syntax:	`class VerifierPublicCtx : public CryptoContext {...};`
Description:	Signature Verification Public key Context interface.

⌋

[SWS_CRYPT_20102] Definition of API function ara::crypto::cryp::AuthCipherCtx::GetDigestService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual DigestService::Uptr GetDigestService () const noexcept=0;`	
Return value:	DigestService::Uptr	Unique smart pointer to DigestService.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get DigestService instance.	

⌋

[SWS_CRYPT_20316] Definition of API function ara::crypto::cryp::AuthCipherCtx::GetDigest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > GetDigest (ReadWriteMemRegion outBuffer, ara::core::Optional< std::size_t > truncationLength) const noexcept=0;`	
Parameters (in):	truncationLength	(Optional) Number of left-most bits to be written to the output buffer
Parameters (out):	outBuffer	Output buffer storing the requested digest fragment or the full digest
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotFinished	--
		if the MAC calculation was not finished by a call of the Finish() method
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the key deployed to this context does not have the kAllowSignature permission
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if outBuffer does not have sufficient capacity to hold the digest
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. Note: If truncation is requested, this function shall not modify bits of the output buffer beyond the left-most truncationLength bits!	

⌋

[SWS_CRYPT_21715] Definition of API function ara::crypto::cryp::AuthCipherCtx::GetTransformation

Status: DRAFT

Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< CryptoTransform > GetTransformation () const noexcept=0;`	
Return value:	ara::core::Result< CryptoTransform >	CryptoTransform
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet
Description:	Get the kind of transformation configured for this context: kEncrypt or kDecrypt.	

⌋

[SWS_CRYPT_20103] Definition of API function ara::crypto::cryp::AuthCipherCtx::GetMaxAssociatedDataSize

Status: DRAFT

Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual std::uint64_t GetMaxAssociatedDataSize () const noexcept=0;`	
Return value:	std::uint64_t	maximal supported size of associated public data in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximal supported size of associated public data.	

⌋

[SWS_CRYPT_23634] Definition of API function ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > ProcessConfidentialData (ReadOnlyMemRegion in, ReadWriteMemRegion out, 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.
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if size of the input buffer is not divisible by the block size (see GetBlockSize())
	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the data processing was not started by a call of the Start() method
	ara::crypto::CryptoErrc::kAuthTagNotValid	--
		if the processed data cannot be authenticated
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Process confidential data and return result. 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.	

⌋

[SWS_CRYPT_23635] Definition of API function ara::crypto::cryp::AuthCipherCtx::ProcessConfidentialData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidInputSize	--
		if size of the input buffer is not divisible by the block size (see Get BlockSize())
	ara::crypto::CryptoErrc::k ProcessingNotStarted	--
		if the data processing was not started by a call of the Start() method
	ara::crypto::CryptoErrc::k AuthTagNotValid	--
		if the processed data cannot be authenticated
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.	

⌋

[SWS_CRYPT_20414] Definition of API function ara::crypto::cryp::AuthCipher Ctx::Reset

Status: DRAFT

Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_23911] Definition of API function ara::crypto::cryp::AuthCipher Ctx::SetKey

Status: DRAFT

Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"
Scope:	`class ara::crypto::cryp::AuthCipherCtx`
Syntax:	`virtual ara::core::Result< void > SetKey (const SymmetricKey &key, CryptoTransform transform) noexcept=0;`

▽

△

Parameters (in):	key	the source key object
	transform	the transformation type "direction indicator"
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if kDecrypt is requested but the provided SymmetricKey cannot be used for decryption (kAllowDataDecryption==false)
	ara::crypto::CryptoErrc::kUsageViolation	--
		if kEncrypt is requested but the provided SymmetricKey cannot be used for encryption (kAllowDataEncryption==false)
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if transform is not kEncrypt or kDecrypt
Description:	Set (deploy) a key to the authenticated cipher symmetric algorithm context.	

⌋

[SWS_CRYPT_24714] Definition of API function ara::crypto::cryp::AuthCipherCtx::Start

Status: DRAFT

Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Start (ara::core::Optional< ReadOnlyMemRegion > iv) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if AuthCipherCtx::SetKey() was never called
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	ara::crypto::CryptoErrc::kUnsupported	--
		if the configured algorithm does not support user provided IV, but an IV is provided
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.	

⌋

[SWS_CRYPT_24715] Definition of API function ara::crypto::cryp::AuthCipherCtx::Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	ara::crypto::CryptoErrc::kUnsupported	--
		if the base algorithm (or its current implementation) principally doesn't support the IV variation
	ara::crypto::CryptoErrc::kUsageViolation	--
		if this transformation type is prohibited by the "allowed usage" restrictions of the provided SecretSeed object
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.	

⌋

[SWS_CRYPT_20312] Definition of API function ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< void > UpdateAssociatedData (const RestrictedUseObject &in) noexcept=0;`	
Parameters (in):	in	a part of input message that should be processed
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--

▽

△

		if the digest calculation was not initiated by a call of the Start() method
	ara::crypto::CryptoErrc::kInvalidUsageOrder	--
		if ProcessConfidentialData has already been called
Description:	Update the digest calculation by the specified RestrictedUseObject. This method is dedicated for cases then the `RestrictedUseObject` is a part of the "message".	

⌋

[SWS_CRYPT_20313] Definition of API function ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< void > UpdateAssociatedData (ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	a part of the input message that should be processed
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
	ara::crypto::CryptoErrc::kInvalidUsageOrder	--
		if ProcessConfidentialData has already been called
Description:	Update the digest calculation by a new chunk of associated data.	

⌋

[SWS_CRYPT_20314] Definition of API function ara::crypto::cryp::AuthCipherCtx::UpdateAssociatedData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::AuthCipherCtx`	
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	-- if the digest calculation was not initiated by a call of the Start() method
	ara::crypto::CryptoErrc::kInvalidUsageOrder	-- if ProcessConfidentialData has already been called
Description:	Update the digest calculation by the specified Byte. This method is convenient for processing of constant tags.	

⌋

[SWS_CRYPT_29035] Definition of API function ara::crypto::cryp::BlockService::GetActualIvBitLength

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/block_service.h"	
Scope:	`class ara::crypto::cryp::BlockService`	
Syntax:	`virtual std::size_t GetActualIvBitLength (ara::core::Optional< CryptoObjectUid > ivUid) const 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Get actual bit-length of an IV loaded to the context.	

⌋

[SWS_CRYPT_29033] Definition of API function ara::crypto::cryp::BlockService::GetBlockSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/block_service.h"	
Scope:	`class ara::crypto::cryp::BlockService`	
Syntax:	`virtual std::size_t GetBlockSize () const noexcept=0;`	
Return value:	std::size_t	size of the block in bytes

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Get block (or internal buffer) size of the base algorithm.

⌋

[SWS_CRYPT_29032] Definition of API function ara::crypto::cryp::BlockService::GetIvSize
Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/block_service.h"	
Scope:	`class ara::crypto::cryp::BlockService`	
Syntax:	`virtual std::size_t GetIvSize () const noexcept=0;`	
Return value:	std::size_t	default expected size of IV in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get default expected size of the Initialization Vector (IV) or nonce.	

⌋

[SWS_CRYPT_29034] Definition of API function ara::crypto::cryp::BlockService::IsValidIvSize
Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/block_service.h"	
Scope:	`class ara::crypto::cryp::BlockService`	
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Verify validity of specific Initialization Vector (IV) length.	

⌋

[SWS_CRYPT_20401] Definition of API function ara::crypto::cryp::CryptoContext::~CryptoContext

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Scope:	`class ara::crypto::cryp::CryptoContext`
Syntax:	`virtual ~CryptoContext () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_20411] Definition of API function ara::crypto::cryp::CryptoContext::GetCryptoPrimitiveId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Scope:	`class ara::crypto::cryp::CryptoContext`	
Syntax:	`virtual CryptoPrimitiveId::Uptr GetCryptoPrimitiveId () const noexcept=0;`	
Return value:	CryptoPrimitiveId::Uptr	Unique smart pointer to CryptoPrimitivId instance containing instance identification.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	get CryptoPrimitivId instance containing instance identification.	

⌋

[SWS_CRYPT_20412] Definition of API function ara::crypto::cryp::CryptoContext::IsInitialized

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Scope:	`class ara::crypto::cryp::CryptoContext`
Syntax:	`virtual bool IsInitialized () const noexcept=0;`

▽

△

Return value:	bool	true if the crypto context is completely initialized and ready to use, and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check if the crypto context is already initialized and ready to use. It checks all required values, including: key value, IV/seed, etc.	

⌋

[SWS_CRYPT_30214] Definition of API function ara::crypto::cryp::CryptoContext::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Scope:	`class ara::crypto::cryp::CryptoContext`
Syntax:	`CryptoContext & operator= (const CryptoContext &other)=delete;`
Description:	Copy-assign another CryptoContext to this instance.

⌋

[SWS_CRYPT_30215] Definition of API function ara::crypto::cryp::CryptoContext::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Scope:	`class ara::crypto::cryp::CryptoContext`
Syntax:	`CryptoContext & operator= (CryptoContext &&other)=delete;`
Description:	Move-assign another CryptoContext to this instance.

⌋

[SWS_CRYPT_41003] Definition of API function ara::crypto::cryp::CryptoContext::CryptoContext

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Scope:	`class ara::crypto::cryp::CryptoContext`
Syntax:	`CryptoContext (const CryptoContext &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41004] Definition of API function ara::crypto::cryp::CryptoContext::CryptoContext

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Scope:	`class ara::crypto::cryp::CryptoContext`
Syntax:	`CryptoContext (CryptoContext &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_20654] Definition of API function ara::crypto::cryp::CryptoContext::MyProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Scope:	`class ara::crypto::cryp::CryptoContext`	
Syntax:	`virtual CryptoProvider & MyProvider () const noexcept=0;`	
Return value:	CryptoProvider &	a reference to Crypto Provider instance that provides this context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get a reference to Crypto Provider of this context.	

⌋

[SWS_CRYPT_20503] Definition of API function ara::crypto::cryp::CryptoObject::~CryptoObject

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Syntax:	`virtual ~CryptoObject () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_20518] Definition of API function ara::crypto::cryp::CryptoObject::Downcast

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`template <class ConcreteObject>` `static ara::core::Result< typename ConcreteObject::Uptrc > Downcast ( CryptoObject::Uptrc &&object) noexcept;`	
Template param:	ConcreteObject	target type (derived from CryptoObject) for downcasting
Parameters (in):	object	unique smart pointer to the constant generic CryptoObject interface
Return value:	ara::core::Result< typename Concrete Object::Uptrc >	unique smart pointer to downcasted constant interface of specified derived type
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k BadObjectType	--
		if an actual type of the object is not the specified ConcreteObject
Description:	Downcast and move unique smart pointer from the generic `CryptoObject` interface to concrete derived object.	

⌋

[SWS_CRYPT_20505] Definition of API function ara::crypto::cryp::CryptoObject::GetCryptoPrimitiveId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`virtual CryptoPrimitiveId::Uptr GetCryptoPrimitiveId () const noexcept=0;`	
Return value:	CryptoPrimitiveId::Uptr	Unique smart pointer to CryptoPrimitiveId
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Return the CryptoPrimitivId of this CryptoObject.	

⌋

[SWS_CRYPT_20514] Definition of API function ara::crypto::cryp::CryptoObject::GetObjectId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`virtual COIdentifier 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:	exception safe	
Thread Safety:	thread-safe	
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!	

⌋

[SWS_CRYPT_20516] Definition of API function ara::crypto::cryp::CryptoObject::GetPayloadSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`virtual std::size_t GetPayloadSize () const noexcept=0;`	
Return value:	std::size_t	size in bytes of the object's payload required for its storage
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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: `MyProvider().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.	

⌋

[SWS_CRYPT_20515] Definition of API function ara::crypto::cryp::CryptoObject::HasDependence

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`virtual COIdentifier 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:	exception safe	
Thread Safety:	thread-safe	
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`.	

⌋

[SWS_CRYPT_20513] Definition of API function ara::crypto::cryp::CryptoObject::IsExportable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the exportability attribute of the crypto object. An exportable object must have an assigned COUID (see `GetObjectId()`).	

⌋

[SWS_CRYPT_20512] Definition of API function ara::crypto::cryp::CryptoObject::IsSession

Status: DRAFT
Upstream requirements: RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
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()`).	

⌋

[SWS_CRYPT_20517] Definition of API function ara::crypto::cryp::CryptoObject::Save

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Scope:	`class ara::crypto::cryp::CryptoObject`	
Syntax:	`virtual ara::core::Result< void > Save (IOInterface &container) const noexcept=0;`	
Parameters (in):	container	IOInterface representing underlying storage
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the object is "session", but the IOInterface represents a KeySlot.
	ara::crypto::CryptoErrc::kContentRestrictions	--
		if the object doesn't satisfy the slot restrictions (keys::KeySlotPrototypeProps)
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if the capacity of the target container is not enough, i.e. if (container.Capacity() <this->StorageSize())
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::kUnreservedResource	--
		if the IOInterface is not opened writeable.
Description:	Save itself to provided IOInterface A CryptoObject with property "session" cannot be saved in a KeySlot.	

⌋

[SWS_CRYPT_30208] Definition of API function ara::crypto::cryp::CryptoObject::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02009

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Syntax:	`CryptoObject & operator= (const CryptoObject &other)=delete;`
Description:	Copy-assign another CryptoObject to this instance.

⌋

[SWS_CRYPT_30209] Definition of API function ara::crypto::cryp::CryptoObject::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Syntax:	`CryptoObject & operator= (CryptoObject &&other)=delete;`
Description:	Move-assign another CryptoObject to this instance.

⌋

[SWS_CRYPT_41001] Definition of API function ara::crypto::cryp::CryptoObject::CryptoObject

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Syntax:	`CryptoObject (const CryptoObject &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41002] Definition of API function ara::crypto::cryp::CryptoObject::CryptoObject

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Syntax:	`CryptoObject (CryptoObject &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_10808] Definition of API function ara::crypto::cryp::CryptoPrimitiveId::~CryptoPrimitiveId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Syntax:	`virtual ~CryptoPrimitiveId () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_20652] Definition of API function ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"	
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`	
Syntax:	`virtual AlgId GetPrimitiveId () const noexcept=0;`	
Return value:	AlgId	the binary Crypto Primitive ID
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get vendor specific ID of the primitive.	

⌋

[SWS_CRYPT_20651] Definition of API function ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveName

Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Syntax:	`virtual const ara::core::StringView GetPrimitiveName () const noexcept=0;`

▽

△

Return value:	const ara::core::String View	the unified name of the crypto primitive
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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!	

⌋

[SWS_CRYPT_30212] Definition of API function ara::crypto::cryp::CryptoPrimitiveId::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Syntax:	`CryptoPrimitiveId & operator= (const CryptoPrimitiveId &other)=delete;`
Description:	Copy-assign another CryptoPrimitiveId to this instance.

⌋

[SWS_CRYPT_30213] Definition of API function ara::crypto::cryp::CryptoPrimitiveId::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Syntax:	`CryptoPrimitiveId & operator= (CryptoPrimitiveId &&other)=delete;`
Description:	Move-assign another CryptoPrimitiveId to this instance.

⌋

[SWS_CRYPT_41017] Definition of API function ara::crypto::cryp::CryptoPrimitiveId::CryptoPrimitiveId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Syntax:	`CryptoPrimitiveId (const CryptoPrimitiveId &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41018] Definition of API function ara::crypto::cryp::CryptoPrimitiveId::CryptoPrimitiveId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Syntax:	`CryptoPrimitiveId (CryptoPrimitiveId &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_20726] Definition of API function ara::crypto::cryp::CryptoProvider::AllocVolatileContainer

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005, RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< VolatileTrustedContainer::Uptr > AllocVolatileContainer (std::size_t capacity) 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:	exception safe	
Thread Safety:	thread-safe	

▽

△

Errors:	ara::crypto::CryptoErrc::k InsufficientCapacity	--
		if the requested capacity cannot be allocated by this CryptoProvider
	ara::crypto::CryptoErrc::k InvalidArgument	--
		if the requested capacity is outside a CryptoProvider specific range
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. A few volatile (temporary) containers can coexist at same time without any affecting each-other.	

⌋

[SWS_CRYPT_20727] Definition of API function ara::crypto::cryp::CryptoProvider::AllocVolatileContainer

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005, RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< VolatileTrustedContainer::Uptr > AllocVolatileContainer (std::pair< 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidArgument	--
		if unsupported combination of object type and algorithm ID presents in the list
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.	

⌋

[SWS_CRYPT_20711] Definition of API function ara::crypto::cryp::CryptoProvider::ConvertToAlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`

▽

△

Syntax:	`virtual ara::core::Result< AlgId > ConvertToAlgId (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:	ara::core::Result< AlgId >	vendor specific binary algorithm ID.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if a primitive with the provided name is not supported
Description:	Convert a common name of crypto algorithm to a correspondent vendor specific binary algorithm ID.	

⌋

[SWS_CRYPT_20712] Definition of API function ara::crypto::cryp::CryptoProvider::ConvertToAlgName

Status: DRAFT
Upstream requirements: RS_CRYPTO_02308

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< ara::core::String > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Convert a vendor specific binary algorithm ID to a correspondent common name of the crypto algorithm.	

⌋

[SWS_CRYPT_30098] Definition of API function ara::crypto::cryp::CryptoProvider::GenerateRandomData

Status: DRAFT

Upstream requirements: RS_CRYPTO_02206

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`ara::core::Result< void > GenerateRandomData (ReadWriteMemRegion out) noexcept;`	
Parameters (out):	out	Buffer to hold the generated random data
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	Thread-safe	
Errors:	ara::crypto::CryptoErrc::kBusyResource	--
		if the requested number of random bytes temporarily cannot be provided; for example due to insufficient entropy or because the resource is in use by another application.
	ara::crypto::CryptoErrc::kUnsupported	--
		if the function is not supported by the CryptoProvider
Description:	Fill the provided output buffer with random data generated from the default random data source of this CryptoProvider.	

⌋

[SWS_CRYPT_20745] Definition of API function ara::crypto::cryp::CryptoProvider::CreateAuthCipherCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02207, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< AuthCipherCtx::Uptr > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from symmetric authenticated stream cipher
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a symmetric authenticated cipher context.	

⌋

[SWS_CRYPT_20751] Definition of API function ara::crypto::cryp::CryptoProvider::CreateDecryptorPrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< DecryptorPrivateCtx::Uptr > CreateDecryptorPrivateCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric encryption/decryption algorithm
Return value:	ara::core::Result< DecryptorPrivateCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a decryption private key context.	

⌋

[SWS_CRYPT_20750] Definition of API function ara::crypto::cryp::CryptoProvider::CreateEncryptorPublicCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< EncryptorPublicCtx::Uptr > CreateEncryptorPublicCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric encryption/decryption algorithm
Return value:	ara::core::Result< EncryptorPublicCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value

▽

△

Description:	Create an encryption public key context.

⌋

[SWS_CRYPT_20747] Definition of API function ara::crypto::cryp::CryptoProvider::CreateHashFunctionCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02205, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< HashFunctionCtx::Uptr > CreateHashFunctionCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
Return value:	ara::core::Result< HashFunctionCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from hash function
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a hash function context.	

⌋

[SWS_CRYPT_20758] Definition of API function ara::crypto::cryp::CryptoProvider::CreateKeyAgreementPrivateCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02104, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< KeyAgreementPrivateCtx::Uptr > CreateKeyAgreementPrivateCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target key-agreement crypto algorithm
Return value:	ara::core::Result< KeyAgreementPrivateCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from key-agreement
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a key-agreement private key context.	

⌋

[SWS_CRYPT_20753] Definition of API function ara::crypto::cryp::CryptoProvider::CreateKeyDecapsulatorPrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104, RS_CRYPTO_02209, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< KeyDecapsulatorPrivateCtx::Uptr > CreateKeyDecapsulatorPrivateCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target KEM crypto algorithm
Return value:	ara::core::Result< KeyDecapsulatorPrivateCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from asymmetric KEM
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a key-decapsulator private key context of a Key Encapsulation Mechanism (KEM).	

⌋

[SWS_CRYPT_20748] Definition of API function ara::crypto::cryp::CryptoProvider::CreateKeyDerivationFunctionCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02103, RS_AP_00144

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`virtual ara::core::Result< KeyDerivationFunctionCtx::Uptr > CreateKeyDerivationFunctionCtx (AlgId algId) noexcept=0;`

▽

△

Parameters (in):	algId	identifier of the target crypto algorithm
Return value:	ara::core::Result< Key DerivationFunction Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidArgument	--
		if algId argument specifies a crypto algorithm different from key derivation function
	ara::crypto::CryptoErrc::k UnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a key derivation function context.	

⌋

[SWS_CRYPT_20752] Definition of API function ara::crypto::cryp::CryptoProvider::CreateKeyEncapsulatorPublicCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02104, RS_CRYPTO_02209, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< KeyEncapsulatorPublicCtx::Uptr > CreateKeyEncapsulatorPublicCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target KEM crypto algorithm
Return value:	ara::core::Result< Key EncapsulatorPublic Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidArgument	--
		if algId argument specifies a crypto algorithm different from asymmetric KEM
	ara::crypto::CryptoErrc::k UnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a key-encapsulator public key context of a Key Encapsulation Mechanism (KEM).	

⌋

[SWS_CRYPT_20746] Definition of API function ara::crypto::cryp::CryptoProvider::CreateMessageAuthnCodeCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02203, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< MessageAuthnCodeCtx::Uptr > CreateMessageAuthnCodeCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
Return value:	ara::core::Result< MessageAuthnCodeCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from symmetric message authentication code
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a symmetric message authentication code context.	

⌋

[SWS_CRYPT_20755] Definition of API function ara::crypto::cryp::CryptoProvider::CreateMsgRecoveryPublicCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202, RS_CRYPTO_02204, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< MsgRecoveryPublicCtx::Uptr > CreateMsgRecoveryPublicCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric crypto algorithm
Return value:	ara::core::Result< MsgRecoveryPublicCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value

▽

△

Description:	Create a message recovery public key context.

⌋

[SWS_CRYPT_20741] Definition of API function ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02206

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< RandomGeneratorCtx::Uptr > CreateRandomGeneratorCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of target RNG algorithm.
Return value:	ara::core::Result< RandomGeneratorCtx::Uptr >	unique smart pointer to the created RNG context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
	ara::crypto::CryptoErrc::kBusyResource	--
		if the context currently cannot be seeded (e.g., due to a lack of entropy)
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the provided AlgId is supported but does not refer to random number generation
Description:	Create a Random Number Generator (RNG) context.	

⌋

[SWS_CRYPT_20754] Definition of API function ara::crypto::cryp::CryptoProvider::CreateSigEncodePrivateCtx

Status: DRAFT

Upstream requirements: RS_CRYPTO_02202, RS_CRYPTO_02204, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< SigEncodePrivateCtx::Uptr > CreateSigEncodePrivateCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric crypto algorithm

▽

△

Return value:	ara::core::Result< SigEncodePrivateCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a signature encoding private key context.	

⌋

[SWS_CRYPT_20756] Definition of API function ara::crypto::cryp::CryptoProvider::CreateSignerPrivateCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< SignerPrivateCtx::Uptr > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from private key signature
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a signature private key context.	

⌋

[SWS_CRYPT_20744] Definition of API function ara::crypto::cryp::CryptoProvider::CreateStreamCipherCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< StreamCipherCtx::Uptr > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from symmetric stream cipher
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a symmetric stream cipher context.	

⌋

[SWS_CRYPT_20742] Definition of API function ara::crypto::cryp::CryptoProvider::CreateSymmetricBlockCipherCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< SymmetricBlockCipherCtx::Uptr > CreateSymmetricBlockCipherCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
Return value:	ara::core::Result< SymmetricBlockCipherCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a symmetric block cipher context.	

⌋

[SWS_CRYPT_20743] Definition of API function ara::crypto::cryp::CryptoProvider::CreateSymmetricKeyWrapperCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104, RS_CRYPTO_02208

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< SymmetricKeyWrapperCtx::Uptr > CreateSymmetricKeyWrapperCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
Return value:	ara::core::Result< SymmetricKeyWrapperCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from symmetric key-wrapping
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value
Description:	Create a symmetric key-wrap algorithm context.	

⌋

[SWS_CRYPT_20757] Definition of API function ara::crypto::cryp::CryptoProvider::CreateVerifierPublicCtx

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204, RS_AP_00144

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< VerifierPublicCtx::Uptr > CreateVerifierPublicCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target signature crypto algorithm
Return value:	ara::core::Result< VerifierPublicCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if algId argument specifies a crypto algorithm different from public key signature verification
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId argument has an unsupported value

▽

△

Description:	Create a signature verification public key context.

⌋

[SWS_CRYPT_20710] Definition of API function ara::crypto::cryp::CryptoProvider::~CryptoProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02107

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`virtual ~CryptoProvider () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_20731] Definition of API function ara::crypto::cryp::CryptoProvider::ExportPublicObject

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105, RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< std::size_t > ExportPublicObject (ReadWriteMemRegion outBuffer, const IOInterface &container, Serializable::FormatId formatId) noexcept=0;`	
Parameters (in):	container	The IOInterface that contains an object for export
	formatId	The CryptoProvider specific identifier of the output format
Parameters (out):	outBuffer	An output buffer large enough to hold the exported object
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kEmptyContainer	--
		if the container is empty
	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if the container contains a secret crypto object

▽

△

	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if outBuffer does not have sufficient capacity
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
Description:	Export publicly an object from a IOInterface (i.e. without an intermediate creation of a crypto object).	

⌋

[SWS_CRYPT_20728] Definition of API function ara::crypto::cryp::CryptoProvider::ExportSecuredObject

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105, RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< std::size_t > ExportSecuredObject (const CryptoObject &object, SymmetricKeyWrapperCtx &transportContext, ReadWriteMemRegion out) noexcept=0;`	
Parameters (in):	object	the crypto object for export
	transportContext	the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyExporting)
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the object cannot be exported due to IsExportable() returning flase
	ara::crypto::CryptoErrc::kIncompleteArgState	--
		if the transportContext is not initialized
	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
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!	

⌋

[SWS_CRYPT_20729] Definition of API function ara::crypto::cryp::CryptoProvider::ExportSecuredObject

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105, RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< std::size_t > ExportSecuredObject (const IOInterface &container, SymmetricKeyWrapperCtx &transportContext, ReadWriteMemRegion out) 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)
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kEmptyContainer	--
		if the container is empty
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if size of the serialized buffer is not enough for saving the output data
	ara::crypto::CryptoErrc::kIncompleteArgState	--
		if the transportContext is not initialized
	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Export securely an object directly from an IOInterface (i.e. without an intermediate creation of a crypto object). This method can be used for re-exporting of just imported object but on another transport key.	

⌋

[SWS_CRYPT_20722] Definition of API function ara::crypto::cryp::CryptoProvider::GeneratePrivateKey

Status: DRAFT

Upstream requirements: RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< PrivateKey::Uptrc > GeneratePrivateKey (AlgId algId, AllowedUsageFlags allowedUsage) 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)
Return value:	ara::core::Result< PrivateKey::Uptrc >	smart unique pointer to the created PrivateKey
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId has an unsupported value
	ara::crypto::CryptoErrc::kIncompatibleArguments	--
		if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)
Description:	Generates an asymmetric key-pair according to the algorithm specified. 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!	

⌋

[SWS_CRYPT_20723] Definition of API function ara::crypto::cryp::CryptoProvider::GenerateSeed

Status: DRAFT

Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< SecretSeed::Uptrc > GenerateSeed (AlgId algId, SecretSeed::Usage allowedUsage) noexcept=0;`	
Parameters (in):	algId	the identifier of target crypto algorithm
	allowedUsage	the flags that define a list of allowed transformation types and ways in which the target seed can be used (see constants in scope of RestrictedUseObject)
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to generated SecretSeed object

▽

△

Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId has an unsupported value
	ara::crypto::CryptoErrc::kIncompatibleArguments	--
		if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)
Description:	Generate a random Secret Seed object according to the algorithm specified.	

⌋

[SWS_CRYPT_20721] Definition of API function ara::crypto::cryp::CryptoProvider::GenerateSymmetricKey

Status: DRAFT

Upstream requirements: RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_-CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_-CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< SymmetricKey::Uptrc > GenerateSymmetricKey (AlgId algId, AllowedUsageFlags allowedUsage) 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)
Return value:	ara::core::Result< SymmetricKey::Uptrc >	smart unique pointer to the created SymmetricKey
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if algId has an unsupported value
	ara::crypto::CryptoErrc::kIncompatibleArguments	--
		if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)
Description:	Generates a symmetric key according to the algorithm specified. 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).	

⌋

[SWS_CRYPT_20725] Definition of API function ara::crypto::cryp::CryptoProvider::GetPayloadStorageSize

Status: DRAFT

Upstream requirements: RS_CRYPTO_02005, RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< std::size_t > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if any argument has an unsupported value
	ara::crypto::CryptoErrc::kIncompatibleArguments	--
		if the arguments are incompatible
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.	

⌋

[SWS_CRYPT_41021] Definition of API function ara::crypto::cryp::CryptoProvider::GetProviderID

Status: DRAFT

Upstream requirements: RS_CRYPTO_02305

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< void > GetProviderID (ara::core::Optional< ReadOnlyMemRegion > in, ReadWriteMemRegion idData) noexcept=0;`	
Parameters (in):	in	Optional input data required by some identification protocols (e.g. SHE/CMD_GET_ID)
Parameters (out):	idData	the identification data of the provider
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	Thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupported	--
		if this provider does not support identification

▽

△

	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if the provided ReadWriteMemRegion parameter idData is insufficient in size to hold the identification data
Description:	Obtain provider specific identification data.	

⌋

[SWS_CRYPT_20724] Definition of API function ara::crypto::cryp::CryptoProvider::GetSerializedSize

Status: DRAFT

Upstream requirements: RS_CRYPTO_02005, RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< std::size_t > GetSerializedSize (CryptoObjectType cryptoObjectType, AlgId algId, Serializable::FormatId formatId) 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, in Bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if any argument has an unsupported value
	ara::crypto::CryptoErrc::kIncompatibleArguments	--
		if any pair of the arguments are incompatible
Description:	Return required buffer size for serialization of an object in specific format.	

⌋

[SWS_CRYPT_20732] Definition of API function ara::crypto::cryp::CryptoProvider::ImportPublicObject

Status: DRAFT

Upstream requirements: RS_CRYPTO_02105, RS_CRYPTO_02112

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`

▽

△

Syntax:	`virtual ara::core::Result< void > ImportPublicObject (IOInterface &container, ReadOnlyMemRegion serialized, ara::core::Optional< CryptoObjectType > expectedObject) noexcept=0;`	
Parameters (in):	serialized	memory region that contains a publicly serialized object
	expectedObject	optional parameter with the expected object type; not providing it or setting it to CryptoObjectType::kUndefined means without check
Parameters (out):	container	IOInterface referencing the memory to store the imported object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if serialized contains incorrect data
	ara::crypto::CryptoErrc::kBadObjectType	--
		if expectedObject is provided and (expectedObject != CryptoObjectType::kUndefined), but the actual object type differs from the expected one
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if capacity of the container is not enough to save the de-serialized object
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::kUnreservedResource	--
		if the IOInterface is not opened writable.
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 is provided and `(expectedObject != CryptoObjectType::kUndefined)` and the actual object type differs from the expected one then this method fails. If the `serialized` contains incorrect data then this method fails.	

⌋

[SWS_CRYPT_20730] Definition of API function ara::crypto::cryp::CryptoProvider::ImportSecuredObject

Status: DRAFT

Upstream requirements: RS_CRYPTO_02105, RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< std::size_t > ImportSecuredObject (IOInterface &container, ReadOnlyMemRegion serialized, SymmetricKeyWrapperCtx &transportContext, ReadWriteMemRegion response, ara::core::Optional< CryptoObjectType > expectedObject) noexcept=0;`	
Parameters (in):	serialized	Memory region that contains a securely serialized object
	transportContext	Symmetric key wrap context initialized by a transport key with allowed usage flag kAllowKeyImporting set to true

▽

△

	expectedObject	optional parameter with the expected object type; not providing it or setting it to CryptoObjectType::kUnknown means without check
Parameters (out):	container	IOInterface referencing memory to store the imported object
	response	An optional response that is required by some protocols, e.g. messages M4 and M5 as specified by AUTOSAR_TR_Secure HardwareExtensions.
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to response
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k UnexpectedValue	--
		if the serialized contains incorrect data
	ara::crypto::CryptoErrc::k BadObjectType	--
		if expectedObject is provided and (expectedObject != CryptoObject Type::kUnknown), but the actual object type differs from the expected one
	ara::crypto::CryptoErrc::k IncompleteArgState	--
		if the transportContext is not initialized
	ara::crypto::CryptoErrc::k IncompatibleObject	--
		if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)
	ara::crypto::CryptoErrc::k InsufficientCapacity	--
		if capacity of the container is not enough to save the deserialized object
	ara::crypto::CryptoErrc::k InsufficientCapacity	--
		if response does not have sufficient capacity
	ara::crypto::CryptoErrc::k ModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::k UnreservedResource	--
		if the IOInterface is not opened writeable.
Description:	Import securely serialized object to the persistent or volatile storage represented by an IOInterface for following processing.	

⌋

[SWS_CRYPT_20733] Definition of API function ara::crypto::cryp::Crypto Provider::LoadObject

Status: DRAFT

Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02002, RS_CRYPTO_02005, RS_-CRYPTO_02006

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`virtual ara::core::Result< CryptoObject::Uptrc > LoadObject (const IOInterface &container) noexcept=0;`

▽

△

Parameters (in):	container	the IOInterface that contains the crypto object for loading
Return value:	ara::core::Result< Crypto Object::Uptrc >	unique smart pointer to the created object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k EmptyContainer	--
		if the container is empty
	ara::crypto::CryptoErrc::k ResourceFault	--
		if the container content is damaged
	ara::crypto::CryptoErrc::k ModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::k IncompatibleObject	--
		if the underlying resource belongs to another, incompatible Crypto Provider
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.	

⌋

[SWS_CRYPT_20764] Definition of API function ara::crypto::cryp::CryptoProvider::LoadPrivateKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02002

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< PrivateKey::Uptrc > 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:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::k EmptyContainer	--
		if the container is empty
	ara::crypto::CryptoErrc::k ResourceFault	--
		if the container content is damaged
	ara::crypto::CryptoErrc::k ModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::k IncompatibleObject	--
		if the underlying resource belongs to another, incompatible Crypto Provider

▽

△

Description:	Load a private key from the IOInterface provided.

⌋

[SWS_CRYPT_20763] Definition of API function ara::crypto::cryp::CryptoProvider::LoadPublicKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02002

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< PublicKey::Uptrc > 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:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kEmptyContainer	--
		if the container is empty
	ara::crypto::CryptoErrc::kResourceFault	--
		if the container content is damaged
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the underlying resource belongs to another, incompatible Crypto Provider
Description:	Load a public key from the IOInterface provided.	

⌋

[SWS_CRYPT_20765] Definition of API function ara::crypto::cryp::CryptoProvider::LoadSecretSeed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02002

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`virtual ara::core::Result< SecretSeed::Uptrc > 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:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kEmptyContainer	--
		if the container is empty
	ara::crypto::CryptoErrc::kResourceFault	--
		if the container content is damaged
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the underlying resource belongs to another, incompatible Crypto Provider
Description:	Load secret seed from the IOInterface provided.	

⌋

[SWS_CRYPT_20762] Definition of API function ara::crypto::cryp::CryptoProvider::LoadSymmetricKey

Status: DRAFT

Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02002

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Scope:	`class ara::crypto::cryp::CryptoProvider`	
Syntax:	`virtual ara::core::Result< SymmetricKey::Uptrc > LoadSymmetricKey (const IOInterface &container) noexcept=0;`	
Parameters (in):	container	the IOInterface that contains the crypto object for loading
Return value:	ara::core::Result< SymmetricKey::Uptrc >	unique smart pointer to the SymmetricKey
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kEmptyContainer	--
		if the container is empty
	ara::crypto::CryptoErrc::kResourceFault	--
		if the container content is damaged
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the underlying resource belongs to another, incompatible Crypto Provider
Description:	Load a symmetric key from the IOInterface provided.	

⌋

[SWS_CRYPT_29023] Definition of API function ara::crypto::cryp::CryptoService::GetBlockSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_service.h"	
Scope:	`class ara::crypto::cryp::CryptoService`	
Syntax:	`virtual std::size_t GetBlockSize () const noexcept=0;`	
Return value:	std::size_t	size of the block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_29021] Definition of API function ara::crypto::cryp::CryptoService::GetMaxInputSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_service.h"	
Scope:	`class ara::crypto::cryp::CryptoService`	
Syntax:	`virtual std::size_t GetMaxInputSize () const noexcept=0;`	
Return value:	std::size_t	maximum size of the input data block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximum expected size of the input data block. For a context configured with Crypto Transform::kEncrypt this interface returns the block size. For a context configured with Crypto Transform::kDecrypt this interface returns the block size, if the context does not implement padding. Otherwise, it assumes the full block size is used for payload and returns the block size plus the maximum Byte size available for padding.	

⌋

[SWS_CRYPT_29022] Definition of API function ara::crypto::cryp::CryptoService::GetMaxOutputSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/crypto_service.h"	
Scope:	`class ara::crypto::cryp::CryptoService`	
Syntax:	`virtual std::size_t GetMaxOutputSize () const noexcept=0;`	
Return value:	std::size_t	maximum size of the output data block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximum possible size of the output data block. For a context configured with Crypto Transform::kEncrypt this interface returns the block size. For a context configured with Crypto Transform::kDecrypt this interface returns the block size, if the context does not implement padding. Otherwise, it assumes the full block size is used for payload and returns the block size plus the maximum Byte size available for padding.	

⌋

[SWS_CRYPT_30216] Definition of API function ara::crypto::cryp::Crypto Provider::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`CryptoProvider & operator= (const CryptoProvider &other)=delete;`
Description:	Copy-assign another CryptoProvider to this instance.

⌋

[SWS_CRYPT_30217] Definition of API function ara::crypto::cryp::Crypto Provider::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`CryptoProvider & operator= (CryptoProvider &&other)=delete;`

▽

△

Description:	Move-assign another CryptoProvider to this instance.

⌋

[SWS_CRYPT_41005] Definition of API function ara::crypto::cryp::CryptoProvider::CryptoProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`CryptoProvider (const CryptoProvider &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41006] Definition of API function ara::crypto::cryp::CryptoProvider::CryptoProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Syntax:	`CryptoProvider (CryptoProvider &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_20802] Definition of API function ara::crypto::cryp::DecryptorPrivateCtx::GetCryptoService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"	
Scope:	`class ara::crypto::cryp::DecryptorPrivateCtx`	
Syntax:	`virtual CryptoService::Uptr GetCryptoService () const noexcept=0;`	
Return value:	CryptoService::Uptr	Unique smart pointer to CryptoService

▽

△

Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Get CryptoService instance.

⌋

[SWS_CRYPT_20812] Definition of API function ara::crypto::cryp::DecryptorPrivateCtx::ProcessBlock
Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"	
Scope:	`class ara::crypto::cryp::DecryptorPrivateCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > ProcessBlock (ReadOnlyMemRegion in, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	in	The input data block
Parameters (out):	out	The output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if input data is is not exactly on the block size of the configured algorithm
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity to hold the tranformation result
	ara::crypto::CryptoErrc::kUninitializedContext	--
		DecryptorPrivateCtx::SetKey() was never called
Description:	Decrypt an input block according to the configured algorithm. Transformation is done on full blocks of input data without streaming. Hence, a call of ProcessBlock must provide a full block of input data.	

⌋

[SWS_CRYPT_20811] Definition of API function ara::crypto::cryp::DecryptorPrivateCtx::Reset
Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"
Scope:	`class ara::crypto::cryp::DecryptorPrivateCtx`

▽

△

Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_20810] Definition of API function ara::crypto::cryp::DecryptorPrivateCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"	
Scope:	`class ara::crypto::cryp::DecryptorPrivateCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Description:	Set (deploy) a key to the decryptor private algorithm context.	

⌋

[SWS_CRYPT_29013] Definition of API function ara::crypto::cryp::DigestService::Compare

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/digest_service.h"	
Scope:	`class ara::crypto::cryp::DigestService`	
Syntax:	`virtual ara::core::Result< bool > Compare (ReadOnlyMemRegion expected, ara::core::Optional< std::size_t > truncationLength) const noexcept=0;`	
Parameters (in):	expected	The memory region containing an expected digest value

▽

△

	truncationLength	(Optional) Number of left-most bits to be compared
Return value:	ara::core::Result< bool >	true if all or the left-most truncationLength bits of the calculated digest and the expected digest are equal
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotFinished	--
		if the digest calculation was not finished by a call of the Finish() method
	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if truncationLength is provided but greater than the number of bits in expected
Description:	Compare the calculated digest against an expected value. Entire digest value is kept in the context up to next call of `Start()`, therefore any its part can be verified again or extracted. If truncationLength equals 0 or `expected` is empty then return `false`. Note: if truncationLength is greater than the number of calculated bits, only the actual size of the digest can be compared.	

⌋

[SWS_CRYPT_29012] Definition of API function ara::crypto::cryp::DigestService::GetDigestSize
Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/digest_service.h"	
Scope:	`class ara::crypto::cryp::DigestService`	
Syntax:	`virtual std::size_t GetDigestSize () const noexcept=0;`	
Return value:	std::size_t	size of the full output from this digest-function in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the output digest size.	

⌋

[SWS_CRYPT_29015] Definition of API function ara::crypto::cryp::DigestService::IsFinished
Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/digest_service.h"
Scope:	`class ara::crypto::cryp::DigestService`
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check current status of the stream processing: finished or no.	

⌋

[SWS_CRYPT_29014] Definition of API function ara::crypto::cryp::DigestService::IsStarted

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/digest_service.h"	
Scope:	`class ara::crypto::cryp::DigestService`	
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check current status of the stream processing: started or no.	

⌋

[SWS_CRYPT_21002] Definition of API function ara::crypto::cryp::EncryptorPublicCtx::GetCryptoService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Scope:	`class ara::crypto::cryp::EncryptorPublicCtx`	
Syntax:	`virtual CryptoService::Uptr GetCryptoService () const noexcept=0;`	
Return value:	CryptoService::Uptr	Unique smart pointer to CryptoService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get CryptoService instance.	

⌋

[SWS_CRYPT_21012] Definition of API function ara::crypto::cryp::EncryptorPublicCtx::ProcessBlock

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Scope:	`class ara::crypto::cryp::EncryptorPublicCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > ProcessBlock (ReadOnlyMemRegion in, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	in	Input data block
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the context does not support padding and the length of provided input data (in) is less than the block size of the configured algorithm
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if more input data is provided than the block size of the configured algorithm
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity to hold the tranformation result
	ara::crypto::CryptoErrc::kUninitializedContext	--
		if EncryptorPublicCtx::SetKey() was never called
Description:	Encrypt an input block according to the cryptor configuration. Transformation is done on full blocks of input data without streaming. Hence, a call of ProcessBlock must provide sufficient data to fill the block, which means it can never be more than the block size and less only when the context supports padding.	

⌋

[SWS_CRYPT_21011] Definition of API function ara::crypto::cryp::EncryptorPublicCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Scope:	`class ara::crypto::cryp::EncryptorPublicCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Clear the crypto context.

⌋

[SWS_CRYPT_21010] Definition of API function ara::crypto::cryp::EncryptorPublicCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Scope:	`class ara::crypto::cryp::EncryptorPublicCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PublicKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Description:	Set (deploy) a key to the encryptor public algorithm context.	

⌋

[SWS_CRYPT_29041] Definition of API function ara::crypto::cryp::ExtensionService::~ExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/extension_service.h"
Scope:	`class ara::crypto::cryp::ExtensionService`
Syntax:	`virtual ~ExtensionService () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_29045] Definition of API function ara::crypto::cryp::ExtensionService::GetActualKeyBitLength

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Scope:	`class ara::crypto::cryp::ExtensionService`	
Syntax:	`virtual std::size_t GetActualKeyBitLength () const noexcept=0;`	
Return value:	std::size_t	actual length of a key (now set to the algorithm context) in bits
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_29047] Definition of API function ara::crypto::cryp::ExtensionService::GetActualKeyCOUID

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Scope:	`class ara::crypto::cryp::ExtensionService`	
Syntax:	`virtual CryptoObjectUid GetActualKeyCOUID () const noexcept=0;`	
Return value:	CryptoObjectUid	the COUID of the CryptoObject
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_29046] Definition of API function ara::crypto::cryp::ExtensionService::GetAllowedUsage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/extension_service.h"
Scope:	`class ara::crypto::cryp::ExtensionService`

▽

△

Syntax:	`virtual AllowedUsageFlags GetAllowedUsage () const noexcept=0;`	
Return value:	AllowedUsageFlags	a combination of bit-flags that specifies allowed usages of the context
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_29044] Definition of API function ara::crypto::cryp::ExtensionService::GetMaxKeyBitLength

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Scope:	`class ara::crypto::cryp::ExtensionService`	
Syntax:	`virtual std::size_t GetMaxKeyBitLength () const noexcept=0;`	
Return value:	std::size_t	maximal supported length of the key in bits
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximal supported key length in bits.	

⌋

[SWS_CRYPT_29043] Definition of API function ara::crypto::cryp::ExtensionService::GetMinKeyBitLength

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Scope:	`class ara::crypto::cryp::ExtensionService`	
Syntax:	`virtual std::size_t GetMinKeyBitLength () const noexcept=0;`	
Return value:	std::size_t	minimal supported length of the key in bits
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get minimal supported key length in bits.	

⌋

[SWS_CRYPT_29048] Definition of API function ara::crypto::cryp::ExtensionService::IsKeyBitLengthSupported

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Scope:	`class ara::crypto::cryp::ExtensionService`	
Syntax:	`virtual bool IsKeyBitLengthSupported (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:	exception safe	
Thread Safety:	thread-safe	
Description:	Verify supportness of specific key length by the context.	

⌋

[SWS_CRYPT_29049] Definition of API function ara::crypto::cryp::ExtensionService::IsKeyAvailable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Scope:	`class ara::crypto::cryp::ExtensionService`	
Syntax:	`virtual bool IsKeyAvailable () const noexcept=0;`	
Return value:	bool	FALSE if no key has been set
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check if a key has been set to this context.	

⌋

[SWS_CRYPT_30218] Definition of API function ara::crypto::cryp::ExtensionService::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/extension_service.h"
Scope:	`class ara::crypto::cryp::ExtensionService`
Syntax:	`ExtensionService & operator= (const ExtensionService &other)=delete;`
Description:	Copy-assign another ExtensionService to this instance.

⌋

[SWS_CRYPT_30219] Definition of API function ara::crypto::cryp::ExtensionService::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/extension_service.h"
Scope:	`class ara::crypto::cryp::ExtensionService`
Syntax:	`ExtensionService & operator= (ExtensionService &&other)=delete;`
Description:	Move-assign another ExtensionService to this instance.

⌋

[SWS_CRYPT_41007] Definition of API function ara::crypto::cryp::ExtensionService::ExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/extension_service.h"
Scope:	`class ara::crypto::cryp::ExtensionService`
Syntax:	`ExtensionService (const ExtensionService &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41008] Definition of API function ara::crypto::cryp::ExtensionService::ExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/extension_service.h"
Scope:	`class ara::crypto::cryp::ExtensionService`
Syntax:	`ExtensionService (ExtensionService &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_21115] Definition of API function ara::crypto::cryp::HashFunctionCtx::Finish

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302, RS_CRYPTO_02205

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > Finish (ReadWriteMemRegion out) noexcept=0;`	
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if Start() has not been successfully called before.
	ara::crypto::CryptoErrc::kInvalidUsageOrder	--
		if Update() has not been called successfully after the last call to Start()
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Finish the digest calculation and write the result to the provided output buffer. Only after call of this method the digest can be signed, verified, extracted or compared.	

⌋

[SWS_CRYPT_21102] Definition of API function ara::crypto::cryp::HashFunctionCtx::GetDigestService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual DigestService::Uptr GetDigestService () const noexcept=0;`	
Return value:	DigestService::Uptr	Unique smart pointer to DigestService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get DigestService instance.	

⌋

[SWS_CRYPT_21116] Definition of API function ara::crypto::cryp::HashFunctionCtx::GetDigest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > GetDigest (ReadWriteMemRegion out, ara::core::Optional< std::size_t > truncationLength) const noexcept=0;`	
Parameters (in):	truncationLength	(Optional) Number of left-most bits to be written to the output buffer
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotFinished	--
		if the digest calculation was not finished by a call of HashFunctionCtx::Finish()
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity to hold the digest
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 more data is requested than available, only the calculated digest will be written to `out`. Note: in case `truncationLength` is 0, no data is written to `out` and 0 is returned. Note: If truncation is requested, this function shall not modify bits of the output buffer beyond the left-most truncationLength bits!	

⌋

[SWS_CRYPT_21118] Definition of API function ara::crypto::cryp::HashFunctionCtx::Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > Start () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kMissingArgument	--
		the configured hash function expected an IV
Description:	Initialize the context for a new data stream processing or generation (depending on the primitive) without IV.	

⌋

[SWS_CRYPT_21110] Definition of API function ara::crypto::cryp::HashFunctionCtx::Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > Start (ReadOnlyMemRegion iv) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	ara::crypto::CryptoErrc::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)
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.	

⌋

[SWS_CRYPT_21111] Definition of API function ara::crypto::cryp::HashFunctionCtx::Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	ara::crypto::CryptoErrc::kUnsupported	--
		if the base algorithm (or its current implementation) principally doesn't support the IV variation
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.	

⌋

[SWS_CRYPT_21112] Definition of API function ara::crypto::cryp::HashFunctionCtx::Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > Update (const RestrictedUseObject &in) noexcept=0;`	
Parameters (in):	in	a part of input message that should be processed
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
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".	

⌋

[SWS_CRYPT_21113] Definition of API function ara::crypto::cryp::HashFunctionCtx::Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > Update (ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	a part of the input message that should be processed
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
Description:	Update the digest calculation context by a new part of the message.	

⌋

[SWS_CRYPT_21114] Definition of API function ara::crypto::cryp::HashFunctionCtx::Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Scope:	`class ara::crypto::cryp::HashFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
Description:	Update the digest calculation context by a new part of the message. This method is convenient for processing of constant tags.	

⌋

[SWS_CRYPT_21312] Definition of API function ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyAgreementPrivateCtx`	
Syntax:	`virtual ara::core::Result< SymmetricKey::Uptrc > AgreeKey (const PublicKey &otherSideKey, CryptoAlgId targetAlgId, AllowedUsageFlags allowedUsage, ara::core::Optional< ReadOnlyMemRegion > salt, ara::core::Optional< ReadOnlyMemRegion > ctxLabel) const noexcept=0;`	
Parameters (in):	otherSideKey	the public key of the other side of the Key-Agreement
	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< SymmetricKey::Uptrc >	a unique pointer to SymmetricKey object, which contains the computed shared secret or key material produced by the Key-Agreement algorithm
Exception Safety:	exception safe	
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
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.	

⌋

[SWS_CRYPT_21311] Definition of API function ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeSeed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"
Scope:	`class ara::crypto::cryp::KeyAgreementPrivateCtx`
Syntax:	`virtual ara::core::Result< SecretSeed::Uptrc > 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< Secret Seed::Uptrc >	unique pointer to SecretSeed object, which contains the key material produced by the Key-Agreement algorithm
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	CryptoErrc::k UninitializedContext	--
		if the context was not initialized by a key value
	CryptoErrc::k IncompatibleObject	--
		if the public and private keys correspond to different algorithms
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).	

⌋

[SWS_CRYPT_21302] Definition of API function ara::crypto::cryp::KeyAgreementPrivateCtx::GetExtensionService

Status: DRAFT

Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyAgreementPrivateCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get ExtensionService instance.	

⌋

[SWS_CRYPT_21314] Definition of API function ara::crypto::cryp::KeyAgreementPrivateCtx::Reset

Status: DRAFT

Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyAgreementPrivateCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Clear the crypto context.

⌋

[SWS_CRYPT_21313] Definition of API function ara::crypto::cryp::KeyAgreementPrivateCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyAgreementPrivateCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	CryptoErrc::k IncompatibleObject	--
		if the provided key object is incompatible with this private key context
	CryptoErrc::kUsage Violation	--
		if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Description:	Set (deploy) a key to the key agreement private algorithm context.	

⌋

[SWS_CRYPT_21315] Definition of API function ara::crypto::cryp::KeyAgreementPrivateCtx::SetKDF

Status: DRAFT
Upstream requirements: RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyAgreementPrivateCtx`	
Syntax:	`virtual void 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:	exception safe	

▽

△

Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_21412] Definition of API function ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateKey

Status: DRAFT

Upstream requirements: RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`	
Syntax:	`virtual ara::core::Result< SymmetricKey::Uptrc > DecapsulateKey (ReadOnlyMemRegion input, CryptoAlgId keyingDataAlgId, KeyDerivationFunctionCtx &kdf, CryptoAlgId kekAlgId, ara::core::Optional< AllowedUsageFlags > allowedUsage) const noexcept=0;`	
Parameters (in):	input	an input buffer (its size should be equal GetEncapsulatedSize() 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< SymmetricKey::Uptrc >	unique smart pointer of the symmetric key object instantiated from the decapsulated keying data
Exception Safety:	exception safe	
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
	CryptoErrc::kIncompatibleObject	--
		If the decapsulated keying-data is incompatible "shorter or longer" with the target crypto algorithm specified by parameter keyingDataAlgId.
Description:	Decapsulate the keying data to be used for subsequent processing (e.g. secure communication). Produced `SymmetricKey` object has following attributes: session, non-exportable.	

⌋

[SWS_CRYPT_21411] Definition of API function ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateSeed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`	
Syntax:	`virtual ara::core::Result< SecretSeed::Uptrc > 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:	exception safe	
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
Description:	Decapsulate key material. Produced `SecretSeed` object has following attributes: session, non-exportable, AlgID = this KEM AlgID.	

⌋

[SWS_CRYPT_21416] Definition of API function ara::crypto::cryp::KeyDecapsulatorPrivateCtx::GetEncapsulatedSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`	
Syntax:	`virtual std::size_t GetEncapsulatedSize () const noexcept=0;`	
Return value:	std::size_t	size of the encapsulated data block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get fixed size of the encapsulated data block.	

⌋

[SWS_CRYPT_21402] Definition of API function ara::crypto::cryp::KeyDecapsulatorPrivateCtx::GetExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get ExtensionService instance.	

⌋

[SWS_CRYPT_21415] Definition of API function ara::crypto::cryp::KeyDecapsulatorPrivateCtx::GetKekEntropy

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`	
Syntax:	`virtual std::size_t GetKekEntropy () const noexcept=0;`	
Return value:	std::size_t	entropy of the KEK material in bits
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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).	

⌋

[SWS_CRYPT_21414] Definition of API function ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_21413] Definition of API function ara::crypto::cryp::KeyDecapsulatorPrivateCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
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
Description:	Set (deploy) a key to the key decapsulator private algorithm context.	

⌋

[SWS_CRYPT_21512] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::AddSalt

Status: DRAFT

Upstream requirements: RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_-CRYPTO_02111

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Add a salt value stored in a (non-secret) ReadOnlyMemRegion.	

⌋

[SWS_CRYPT_21513] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::AddSecretSalt

Status: DRAFT

Upstream requirements: RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_-CRYPTO_02111

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Add a secret salt value stored in a SecretSeed object.	

⌋

[SWS_CRYPT_21514] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::ConfigIterations

Status: DRAFT

Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual std::uint32_t ConfigIterations (std::uint32_t iterations) 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Configure the number of iterations that will be applied by default. Implementation can restrict minimal and/or maximal value of the iterations number.	

⌋

[SWS_CRYPT_21515] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveKey

Status: DRAFT

Upstream requirements: RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_-CRYPTO_02111, RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< SymmetricKey::Uptrc > DeriveKey (ara::core::Optional< AllowedUsageFlags > allowedUsage) const noexcept=0;`	
Parameters (in):	allowedUsage	Optional allowed usage for the derived key
Return value:	ara::core::Result< SymmetricKey::Uptrc >	unique smart pointer to the created instance of derived symmetric key
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not sufficiently initialized
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the AllowedUsageFlags for derived key-material of the deployed RestrictedUseObject restrict one or more usage flags provided with allowedUsage

▽

△

Description:	Derive a symmetric key from the provided key material and provided context configuration. If `allowedUsage` is provided, it can only further restrict usage of derived key-material based on derived allowed usage by the RestrictedUseObject deployed; otherwise all allowed usage flags are set to false except those for which a corresponding derived allowed usage of the deployed RestrictedUseObject is set to true.

⌋

[SWS_CRYPT_21516] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveSeed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< SecretSeed::Uptrc > DeriveSeed (ara::core::Optional< AllowedUsageFlags > allowedUsage) const noexcept=0;`	
Parameters (in):	allowedUsage	Optional allowed usage for the derived seed
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to the created SecretSeed object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not sufficiently initialized
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the AllowedUsageFlags for derived key-material of the deployed RestrictedUseObject restrict one or more usage flags provided with allowedUsage
Description:	Derive a SecretSeed from the provided key material and provided context configuration. If `allowedUsage` is provided, it can only further restrict usage of derived key-material based on derived allowed usage by the RestrictedUseObject deployed; otherwise all allowed usage flags are set to false except those for which a corresponding derived allowed usage of the deployed RestrictedUseObject is set to true.	

⌋

[SWS_CRYPT_21524] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`

▽

△

Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_21517] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::GetExtensionService

Status: DRAFT

Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get ExtensionService instance.	

⌋

[SWS_CRYPT_21519] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::GetKeyIdSize

Status: DRAFT

Upstream requirements: RS_CRYPTO_02103

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual std::size_t GetKeyIdSize () const noexcept=0;`	
Return value:	std::size_t	size of the key ID in bytes configured by the last call of the Init() call.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_21520] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::GetTargetAlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02103

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< AlgId > GetTargetAlgId () const noexcept=0;`	
Return value:	ara::core::Result< AlgId >	the symmetric algorithm ID of the target key, configured by the last call of the Init() method
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		If the context was not configured yet by a call of the Init() method
Description:	Get the symmetric algorithm ID of target (slave) key.	

⌋

[SWS_CRYPT_21521] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::GetTargetAllowedUsage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual AllowedUsageFlags GetTargetAllowedUsage () const noexcept=0;`	
Return value:	AllowedUsageFlags	allowed key usage bit-flags of target keys
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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, kAllowKdfMaterialAnyUsage shall be returned.	

⌋

[SWS_CRYPT_21522] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::GetTargetKeyBitLength

Status: DRAFT

Upstream requirements: RS_CRYPTO_02103

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual std::size_t GetTargetKeyBitLength () const noexcept=0;`	
Return value:	std::size_t	the length of target (diversified) key in bits
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the bit-length of target (diversified) keys. Returned value is configured by the context factory method, i.e. independent from configuration by the `Init()` calls.	

⌋

[SWS_CRYPT_21523] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::Init

Status: DRAFT

Upstream requirements: RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_-CRYPTO_02111, RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > Init (ReadOnlyMemRegion targetKeyId, AlgId targetAlgId, ara::core::Optional< AllowedUsageFlags > allowedUsage, ara::core::Optional< ReadOnlyMemRegion > ctxLabel) noexcept=0;`	
Parameters (in):	targetKeyId	ID of the target key
	targetAlgId	the identifier of the symmetric crypto algorithm this key shall be used with.
	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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if targetAlgId is not a valid AlgId of a symmetric cryptographic key
	ara::crypto::CryptoErrc::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

▽

△

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.

⌋

[SWS_CRYPT_21525] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::SetSourceKeyMaterial

Status: DRAFT

Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > SetSourceKeyMaterial (const RestrictedUseObject &sourceKM) noexcept=0;`	
Parameters (in):	sourceKM	the source key-material
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if deriving a key is prohibited by the "allowed usage" restrictions of the provided source key-material
	ara::crypto::CryptoErrc::kBruteForceRisk	--
		if key length of the sourceKm is below of an internally defined limitation
Description:	Set (deploy) key-material to the key derivation algorithm context.	

⌋

[SWS_CRYPT_41019] Definition of API function ara::crypto::cryp::KeyDerivationFunctionCtx::SetSourceKeyMaterial

Status: DRAFT

Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`	
Syntax:	`virtual ara::core::Result< void > SetSourceKeyMaterial (const ReadOnlyMemRegion sourceMaterial) noexcept=0;`	
Parameters (in):	sourceMaterial	the source raw-data to be derived into a cryptographic key

▽

△

Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	Thread-safe	
Errors:	ara::crypto::CryptoErrc::k BruteForceRisk	--
		if key length of the sourceMaterial is below of stack internally defined limitation
Description:	Set (deploy) source material to the key derivation algorithm context. This overload captures use cases in which the user application intends to derive a cryptographic key from raw input data (e.g. password) for session usage.	

⌋

[SWS_CRYPT_21818] Definition of API function ara::crypto::cryp::KeyEncapsulatorPublicCtx::GetEncapsulatedSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`	
Syntax:	`virtual std::size_t GetEncapsulatedSize () const noexcept=0;`	
Return value:	std::size_t	size of the encapsulated data block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get fixed size of the encapsulated data block.	

⌋

[SWS_CRYPT_21802] Definition of API function ara::crypto::cryp::KeyEncapsulatorPublicCtx::GetExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get ExtensionService instance.	

⌋

[SWS_CRYPT_21817] Definition of API function ara::crypto::cryp::KeyEncapsulatorPublicCtx::GetKekEntropy

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`	
Syntax:	`virtual std::size_t GetKekEntropy () const noexcept=0;`	
Return value:	std::size_t	entropy of the KEK material in bits
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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).	

⌋

[SWS_CRYPT_21810] Definition of API function ara::crypto::cryp::KeyEncapsulatorPublicCtx::AddKeyingData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`	
Syntax:	`virtual ara::core::Result< void > AddKeyingData (const RestrictedUseObject &keyingData) noexcept=0;`	
Parameters (in):	keyingData	the payload to be protected
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	CryptoErrc::kUsageViolation	--
		if the keyingData cannot be exported due to CryptoObject::IsExportable() returning FALSE
	CryptoErrc::kIncompatibleObject	--
		if the keyingData belongs to a different CryptoProvider
	CryptoErrc::kInvalidInputSize	--
		if this context does not support the size of the keyingData
Description:	Add the content to be encapsulated (payload) according to RFC 5990 ("keying data"). At the moment only SymmetricKey and SecretSeed objects are supported.	

⌋

[SWS_CRYPT_21813] Definition of API function ara::crypto::cryp::KeyEncapsulatorPublicCtx::Encapsulate

Status: DRAFT

Upstream requirements: RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > Encapsulate (KeyDerivationFunctionCtx &kdf, CryptoAlgId kekAlgId, ReadWriteMemRegion out) 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
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
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
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Encapsulate the last set keying-data and write the result to the output buffer.	

⌋

[SWS_CRYPT_21816] Definition of API function ara::crypto::cryp::KeyEncapsulatorPublicCtx::Reset

Status: DRAFT

Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_21815] Definition of API function ara::crypto::cryp::KeyEncapsulatorPublicCtx::SetKey

Status: DRAFT

Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PublicKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
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
Description:	Set (deploy) a key to the key encapsulator public algorithm context.	

⌋

[SWS_CRYPT_22115] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::Finish

Status: DRAFT

Upstream requirements: RS_CRYPTO_02302, RS_CRYPTO_02203

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< void > Finish () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the key deployed to this context does not have the kAllow Signature permission
Description:	Finish the MAC calculation. The MAC can only be verified, extracted or compared after this method has been called.	

⌋

[SWS_CRYPT_22102] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::GetDigestService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual DigestService::Uptr GetDigestService () const noexcept=0;`	
Return value:	DigestService::Uptr	Unique smart pointer to DigestService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get DigestService instance.	

⌋

[SWS_CRYPT_22116] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::GetDigest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > GetDigest (ReadWriteMemRegion out, ara::core::Optional< std::size_t > truncationLength) const noexcept=0;`	
Parameters (in):	truncationLength	(Optional) Number of left-most bits to be written to the output buffer
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotFinished	--
		if the digest calculation was not finished by a call of MessageAuthnCodeCtx::Finish()
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the deployed key does not have the kAllowSignature permission
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity to hold the digest
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. Note: If truncation is requested, this function shall not modify bits of the output buffer beyond the left-most truncationLength bits!	

⌋

[SWS_CRYPT_22120] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_22118] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const SymmetricKey &key, CryptoTransform transform) noexcept=0;`	
Parameters (in):	key	Symmetric key to use
	transform	kMacGenerate or kMacVerify
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if kMacGenerate is requested but the provided SymmetricKey cannot be used for MAC generation (kAllowSignature==false)
	ara::crypto::CryptoErrc::kUsageViolation	--
		if kMacVerify is requested but the provided SymmetricKey cannot be used for MAC verification (kAllowVerification==false)
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if transform is not kMacGenerate or kMacVerify
Description:	Set (deploy) a key to the message authn code algorithm context.	

⌋

[SWS_CRYPT_22110] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< void > Start (ara::core::Optional< ReadOnly MemRegion > iv) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if MessageAuthnCodeCtx::SetKey()was never called
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the provided IV contains less Bytes than required for initialization
	ara::crypto::CryptoErrc::kUnsupported	--
		if the algorithm used to create this context does not support user provided IV, but an IV is provided
Description:	Finish initialization of this MessageAuthnCodeCtx by setting a user provided IV, if supported by the algorithm used to create this context. If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋

[SWS_CRYPT_22111] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< void > Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if MessageAuthnCodeCtx::SetKey()was never called
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the provided IV contains less Bytes than required for initialization

▽

△

	ara::crypto::CryptoErrc::kUnsupported	--
		if the algorithm used to create this context does not support user provided IV, but an IV is provided
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the provided SecretSeed object does not have the allowed usages kAllowSignature or kAllowVerification set to true
Description:	Finish initialization of this MessageAuthnCodeCtx by setting a user provided IV, if supported by the algorithm used to create this context. If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋

[SWS_CRYPT_22112] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< void > Update (const RestrictedUseObject &in) noexcept=0;`	
Parameters (in):	in	a part of input message that should be processed
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
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".	

⌋

[SWS_CRYPT_22113] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`
Syntax:	`virtual ara::core::Result< void > Update (ReadOnlyMemRegion in) noexcept=0;`

▽

△

Parameters (in):	in	a part of the input message that should be processed
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
Description:	Update the digest calculation context by a new part of the message.	

⌋

[SWS_CRYPT_22114] Definition of API function ara::crypto::cryp::MessageAuthnCodeCtx::Update

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`	
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the digest calculation was not initiated by a call of the Start() method
Description:	Update the digest calculation context by a new part of the message. This method is convenient for processing of constant tags.	

⌋

[SWS_CRYPT_22210] Definition of API function ara::crypto::cryp::MsgRecoveryPublicCtx::GetExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Scope:	`class ara::crypto::cryp::MsgRecoveryPublicCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService

▽

△

Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Get ExtensionService instance.

⌋

[SWS_CRYPT_22213] Definition of API function ara::crypto::cryp::MsgRecoveryPublicCtx::GetMaxInputSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Scope:	`class ara::crypto::cryp::MsgRecoveryPublicCtx`	
Syntax:	`virtual std::size_t GetMaxInputSize () const noexcept=0;`	
Return value:	std::size_t	maximum size of the input data block in Bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximum expected size of the input data block.	

⌋

[SWS_CRYPT_22214] Definition of API function ara::crypto::cryp::MsgRecoveryPublicCtx::GetMaxOutputSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Scope:	`class ara::crypto::cryp::MsgRecoveryPublicCtx`	
Syntax:	`virtual std::size_t GetMaxOutputSize () const noexcept=0;`	
Return value:	std::size_t	maximum size of the output data block in Bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximum possible size of the message recovered by DecodeAndVerify().	

⌋

[SWS_CRYPT_22215] Definition of API function ara::crypto::cryp::MsgRecoveryPublicCtx::DecodeAndVerify

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Scope:	`class ara::crypto::cryp::MsgRecoveryPublicCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > DecodeAndVerify (ReadOnlyMemRegion in, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	in	the input data block
Parameters (out):	out	buffer to hold the recovered message
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the input does not match the required size for the defined algorithm.
	ara::crypto::CryptoErrc::kUninitializedContext	--
		if MsgRecoveryPublicCtx::SetKey() was never called
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity to hold the recovered message
	ara::crypto::CryptoErrc::kAuthTagNotValid	--
		if verification failed (hashed message does not match extracted message-hash)
Description:	Execute message recovery and signature verification according to the algorithm used to create this context. First, the input buffer is decrypted using the public key, then message and message-hash are extracted. Finally, the message is hashed and the result compared against the extracted message-hash.	

⌋

[SWS_CRYPT_22212] Definition of API function ara::crypto::cryp::MsgRecoveryPublicCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Scope:	`class ara::crypto::cryp::MsgRecoveryPublicCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Clear the crypto context.

⌋

[SWS_CRYPT_22211] Definition of API function ara::crypto::cryp::MsgRecoveryPublicCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Scope:	`class ara::crypto::cryp::MsgRecoveryPublicCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PublicKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Description:	Set (deploy) a key to the msg recovery public algorithm context.	

⌋

[SWS_CRYPT_22511] Definition of API function ara::crypto::cryp::PrivateKey::GetPublicKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108, RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/private_key.h"	
Scope:	`class ara::crypto::cryp::PrivateKey`	
Syntax:	`virtual ara::core::Result< PublicKey::Uptrc > 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the public key correspondent to this private key.	

⌋

[SWS_CRYPT_22711] Definition of API function ara::crypto::cryp::PublicKey::CheckKey

Status: OBSOLETE
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"	
Scope:	`class ara::crypto::cryp::PublicKey`	
Syntax:	`virtual bool CheckKey (bool strongCheck=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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check the key for its correctness.	

⌋

[SWS_CRYPT_22712] Definition of API function ara::crypto::cryp::PublicKey::HashPublicKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"	
Scope:	`class ara::crypto::cryp::PublicKey`	
Syntax:	`virtual ara::core::Result< std::size_t > HashPublicKey (ReadWriteMemRegion out, HashFunctionCtx &hashFunc) const noexcept=0;`	
Parameters (in):	hashFunc	a hash-function instance that should be used for hashing
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if size of the hash buffer is not enough for storing of the result
	ara::crypto::CryptoErrc::kIncompleteArgState	--
		if the hashFunc context is not initialized
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Calculate hash of the Public Key value and write hash to the output buffer. The original public key value BLOB is available via the `Serializable` interface.	

⌋

[SWS_CRYPT_22914] Definition of API function ara::crypto::cryp::RandomGeneratorCtx::AddEntropy

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Scope:	`class ara::crypto::cryp::RandomGeneratorCtx`	
Syntax:	`virtual ara::core::Result< void > AddEntropy (ReadOnlyMemRegion entropy) noexcept=0;`	
Parameters (in):	entropy	a memory region with the additional entropy value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupported	--
		if the RandomGeneratorContext does not support adding entropy.
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.	

⌋

[SWS_CRYPT_22915] Definition of API function ara::crypto::cryp::RandomGeneratorCtx::Generate

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Scope:	`class ara::crypto::cryp::RandomGeneratorCtx`	
Syntax:	`virtual ara::core::Result< void > Generate (ReadWriteMemRegion out) noexcept=0;`	
Parameters (out):	out	Output buffer (as span)
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		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.
	ara::crypto::CryptoErrc::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
Description:	Fill the provided output buffer with a generated random sequence.	

⌋

[SWS_CRYPT_22902] Definition of API function ara::crypto::cryp::RandomGeneratorCtx::GetExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Scope:	`class ara::crypto::cryp::RandomGeneratorCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get ExtensionService instance.	

⌋

[SWS_CRYPT_22911] Definition of API function ara::crypto::cryp::RandomGeneratorCtx::Seed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Scope:	`class ara::crypto::cryp::RandomGeneratorCtx`	
Syntax:	`virtual ara::core::Result< void > Seed (ReadOnlyMemRegion seed) noexcept=0;`	
Parameters (in):	seed	a memory region with the seed value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupported	--
		if the RandomGeneratorContext does not support seeding.
Description:	Set the internal state of the RNG using the provided seed.	

⌋

[SWS_CRYPT_22912] Definition of API function ara::crypto::cryp::RandomGeneratorCtx::Seed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Scope:	`class ara::crypto::cryp::RandomGeneratorCtx`	
Syntax:	`virtual ara::core::Result< void > Seed (const SecretSeed &seed) noexcept=0;`	
Parameters (in):	seed	a memory region with the seed value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUsageViolation	--
		if the provided SecretSeed is not allowed to be used for seeding.
	ara::crypto::CryptoErrc::kUnsupported	--
		if the RandomGeneratorContext does not support seeding.
Description:	Set the internal state of the RNG using the provided seed.	

⌋

[SWS_CRYPT_22913] Definition of API function ara::crypto::cryp::RandomGeneratorCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Scope:	`class ara::crypto::cryp::RandomGeneratorCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const SymmetricKey &key) noexcept=0;`	
Parameters (in):	key	a SymmetricKey with the key used as seed value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUsageViolation	--
		if the provided SymmetricKey is not allowed to be used for seeding.
	ara::crypto::CryptoErrc::kUnsupported	--
		if the RandomGeneratorContext does not support seeding by secret key-material.
Description:	Set the internal state of the RNG using the provided seed.	

⌋

[SWS_CRYPT_24811] Definition of API function ara::crypto::cryp::RestrictedUseObject::GetAllowedUsage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/restricted_use_object.h"	
Scope:	`class ara::crypto::cryp::RestrictedUseObject`	
Syntax:	`virtual Usage GetAllowedUsage () const noexcept=0;`	
Return value:	Usage	a combination of bit-flags that specifies allowed applications of the object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get allowed usages of this object.	

⌋

[SWS_CRYPT_23011] Definition of API function ara::crypto::cryp::SecretSeed::Clone

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Scope:	`class ara::crypto::cryp::SecretSeed`	
Syntax:	`virtual ara::core::Result< SecretSeed::Uptr > Clone (ara::core::Optional< ReadOnlyMemRegion > xorDelta) 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 SecretSeed object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_23012] Definition of API function ara::crypto::cryp::SecretSeed::JumpFrom

Status: OBSOLETE
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Scope:	`class ara::crypto::cryp::SecretSeed`	
Syntax:	`virtual ara::core::Result< void > JumpFrom (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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if this object and the from argument are associated with incompatible cryptographic algorithms
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if value size of the from seed is less then value size of this one
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.	

⌋

[SWS_CRYPT_23014] Definition of API function ara::crypto::cryp::SecretSeed::Jump

Status: OBSOLETE
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Scope:	`class ara::crypto::cryp::SecretSeed`	
Syntax:	`virtual 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:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_23013] Definition of API function ara::crypto::cryp::SecretSeed::Next

Status: OBSOLETE
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Scope:	`class ara::crypto::cryp::SecretSeed`	
Syntax:	`virtual SecretSeed & Next () noexcept=0;`	
Return value:	SecretSeed &	reference to this updated object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Set next value of the secret seed according to "counting" 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).	

⌋

[SWS_CRYPT_23015] Definition of API function ara::crypto::cryp::SecretSeed::operator^=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Scope:	`class ara::crypto::cryp::SecretSeed`	
Syntax:	`virtual SecretSeed & operator^= (const SecretSeed &source) noexcept=0;`	
Parameters (in):	source	right argument for the XOR operation
Return value:	SecretSeed &	reference to this updated object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_23016] Definition of API function ara::crypto::cryp::SecretSeed::operatorˆ=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Scope:	`class ara::crypto::cryp::SecretSeed`	
Syntax:	`virtual SecretSeed & operatorˆ= (ReadOnlyMemRegion source) noexcept=0;`	
Parameters (in):	source	right argument for the XOR operation
Return value:	SecretSeed &	reference to this updated object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_19906] Definition of API function ara::crypto::CryptoException::CryptoException

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_error_domain.h"	
Scope:	`class ara::crypto::CryptoException`	
Syntax:	`explicit CryptoException (ara::core::ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new CryptoException from an ErrorCode.	

⌋

[SWS_CRYPT_23210] Definition of API function ara::crypto::cryp::SigEncodePrivateCtx::GetExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SigEncodePrivateCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Extension service member class.	

⌋

[SWS_CRYPT_23213] Definition of API function ara::crypto::cryp::SigEncodePrivateCtx::GetMaxInputSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SigEncodePrivateCtx`	
Syntax:	`virtual std::size_t GetMaxInputSize () const noexcept=0;`	
Return value:	std::size_t	maximum size of the input data block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximum allowed size of the message that can be encoded into the signature.	

⌋

[SWS_CRYPT_23214] Definition of API function ara::crypto::cryp::SigEncodePrivateCtx::GetMaxOutputSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Scope:	`class ara::crypto::cryp::SigEncodePrivateCtx`
Syntax:	`virtual std::size_t GetMaxOutputSize () const noexcept=0;`

▽

△

Return value:	std::size_t	maximum size of the output data block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns the block size of the cipher.	

⌋

[SWS_CRYPT_23215] Definition of API function ara::crypto::cryp::SigEncodePrivateCtx::SignAndEncode

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SigEncodePrivateCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > SignAndEncode (ReadOnlyMemRegion in, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	in	the input data block
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the input is not match with the required size for the defined algorithm.
	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by a key value
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Sign a message by encoding it into the generated signature according to the algorithm configured for this context, and write the result into the provided output buffer.	

⌋

[SWS_CRYPT_23212] Definition of API function ara::crypto::cryp::SigEncodePrivateCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Scope:	`class ara::crypto::cryp::SigEncodePrivateCtx`

▽

△

Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_23211] Definition of API function ara::crypto::cryp::SigEncodePrivateCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SigEncodePrivateCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Description:	Set (deploy) a key to the sig encode private algorithm context.	

⌋

[SWS_CRYPT_29003] Definition of API function ara::crypto::cryp::SignatureService::GetRequiredHashAlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signature_service.h"	
Scope:	`class ara::crypto::cryp::SignatureService`	
Syntax:	`virtual CryptoPrimitiveId::AlgId GetRequiredHashAlgId () const noexcept=0;`	
Return value:	CryptoPrimitiveId::AlgId	required hash algorithm ID

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Get an ID of hash algorithm required by current signature algorithm.

⌋

[SWS_CRYPT_29002] Definition of API function ara::crypto::cryp::SignatureService::GetRequiredHashSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signature_service.h"	
Scope:	`class ara::crypto::cryp::SignatureService`	
Syntax:	`virtual std::size_t GetRequiredHashSize () const noexcept=0;`	
Return value:	std::size_t	required hash size in bytes
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get the hash size required by current signature algorithm.	

⌋

[SWS_CRYPT_29004] Definition of API function ara::crypto::cryp::SignatureService::GetSignatureSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signature_service.h"	
Scope:	`class ara::crypto::cryp::SignatureService`	
Syntax:	`virtual std::size_t GetSignatureSize () const noexcept=0;`	
Return value:	std::size_t	size of the signature value in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get size of the signature value produced and required by the current algorithm.	

⌋

[SWS_CRYPT_23510] Definition of API function ara::crypto::cryp::SignerPrivateCtx::GetSignatureService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`	
Syntax:	`virtual SignatureService::Uptr GetSignatureService () const noexcept=0;`	
Return value:	SignatureService::Uptr	Unique smart pointer to SignatureService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get SignatureService instance.	

⌋

[SWS_CRYPT_23516] Definition of API function ara::crypto::cryp::SignerPrivateCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_23515] Definition of API function ara::crypto::cryp::SignerPrivateCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`

▽

△

Syntax:	`virtual ara::core::Result< void > SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Description:	Set (deploy) a key to the signer private algorithm context.	

⌋

[SWS_CRYPT_23511] Definition of API function ara::crypto::cryp::SignerPrivateCtx::SignPreHashed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > SignPreHashed (const HashFunctionCtx &hashFn, ara::crypto::Serializable::FormatId fmt, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	hashFn	a finalized hash-function context that contains a digest value ready for sign
	fmt	the signature format
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if hash-function algorithm does not comply with the signature algorithm specification of this context
	ara::crypto::CryptoErrc::kProcessingNotFinished	--
		if the method hash.Finish() was not called before the call of this method
	ara::crypto::CryptoErrc::kUninitializedContext	--
		this context was not initialized by a key value
	ara::crypto::CryptoErrc::kUnsupportedFormat	--
		if specified format is not supported

▽

△

Description:	Sign a provided digest value stored in the hash-function context.

⌋

[SWS_CRYPT_23512] Definition of API function ara::crypto::cryp::SignerPrivateCtx::Sign

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > Sign (ReadOnlyMemRegion value, ara::crypto::Serializable::FormatId fmt, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	value	the (pre-)hashed or direct message value that should be signed
	fmt	The format of the signature to be written to the output buffer
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupportedFormat	--
		if specified format is not supported
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the supplied ReadOnlyMemRegion parameter's size is incompatible with the configured signature algorithm.
	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by a key value
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
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).	

⌋

[SWS_CRYPT_23513] Definition of API function ara::crypto::cryp::SignerPrivateCtx::SignPreHashed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > SignPreHashed (AlgId hashAlgId, ReadOnlyMemRegion hashValue, ara::crypto::Serializable::FormatId fmt, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	hashAlgId	hash function algorithm ID
	hashValue	hash function value (resulting digest without any truncations)
	fmt	the signature format
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if hashAlgId algorithm does not comply with the signature algorithm specification of this context
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the supplied ReadOnlyMemRegion parameter's size is incompatible with the configured signature algorithm.
	ara::crypto::CryptoErrc::kUninitializedContext	--
		this context was not initialized by a key value
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
	ara::crypto::CryptoErrc::kUnsupportedFormat	--
		if specified format is not supported
Description:	Sign a directly provided digest value.	

⌋

[SWS_CRYPT_41041] Definition of API function ara::crypto::cryp::SignerPrivateCtx::AddContextData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204, RS_CRYPTO_02006

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`
Syntax:	`virtual ara::core::Result< void > AddContextData (ReadOnlyMemRegion context) const noexcept=0;`

▽

△

Parameters (in):	context	an optional user-supplied "context" (its support depends on concrete algorithm)
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupported	--
		if the configured signature algorithm does not support context data.
Description:	Add context data as raw bytes. Whether or not this data is needed depends on the signature algorithm.	

⌋

[SWS_CRYPT_41027] Definition of API function ara::crypto::cryp::SignerPrivateCtx::GetSerializedSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > GetSerializedSize ( ara::crypto::Serializable::FormatId formatId) const noexcept=0;`	
Parameters (in):	formatId	the output format
Return value:	ara::core::Result< std::size_t >	the number of Bytes required to hold the signature
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupportedFormat	--
		if the specified format ID is not supported for this object type
Description:	This interface shall return the size of the output buffer required to hold the serialized data of the signature in the requested format.	

⌋

[SWS_CRYPT_23620] Definition of API function ara::crypto::cryp::StreamCipherCtx::CountBytesInCache

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Scope:	`class ara::crypto::cryp::StreamCipherCtx`
Syntax:	`virtual std::size_t CountBytesInCache () const noexcept=0;`

▽

△

Return value:	std::size_t	number of bytes now kept in the context cache
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_23621] Definition of API function ara::crypto::cryp::StreamCipherCtx::EstimateMaxInputSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`std::size_t 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Estimate maximal number of input bytes that may be processed for filling of an output buffer without overflow.	

⌋

[SWS_CRYPT_23622] Definition of API function ara::crypto::cryp::StreamCipherCtx::EstimateRequiredCapacity

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`std::size_t EstimateRequiredCapacity (std::size_t inputSize, ara::core::Optional< bool > isFinal) 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:	exception safe	

▽

△

Thread Safety:	thread-safe
Description:	Estimate minimal required capacity of the output buffer, which is enough for saving a result of input data processing.

⌋

[SWS_CRYPT_23618] Definition of API function ara::crypto::cryp::StreamCipherCtx::FinishBytes

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > FinishBytes (ReadOnlyMemRegion in, ReadWriteMemRegion out) noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the context does not support padding and available data is not a multiple of the block-size.
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the transformation direction is kDecrypt and available data is not a multiple of the block-size
	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if data processing was not started by a call of the Start() method
Description:	Process the final part of message (that may be not aligned to the block-size boundary). If `(IsBytewiseMode() == 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)!	

⌋

[SWS_CRYPT_23602] Definition of API function ara::crypto::cryp::StreamCipherCtx::GetBlockService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual BlockService::Uptr GetBlockService () const noexcept=0;`	
Return value:	BlockService::Uptr	Unique smart pointer to BlockService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get BlockService instance.	

⌋

[SWS_CRYPT_23611] Definition of API function ara::crypto::cryp::StreamCipherCtx::IsBytewiseMode

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check the operation mode for the bytewise property.	

⌋

[SWS_CRYPT_23624] Definition of API function ara::crypto::cryp::StreamCipherCtx::GetTransformation
Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< CryptoTransform > GetTransformation () const noexcept=0;`	
Return value:	ara::core::Result< CryptoTransform >	CryptoTransform
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet
Description:	Get the kind of transformation configured for this context: kEncrypt or kDecrypt.	

⌋

[SWS_CRYPT_23612] Definition of API function ara::crypto::cryp::StreamCipherCtx::IsSeekableMode
Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual bool IsSeekableMode () const noexcept=0;`	
Return value:	bool	true the seek operation is supported in the current mode and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check if the seek operation is supported in the current mode.	

⌋

[SWS_CRYPT_23614] Definition of API function ara::crypto::cryp::StreamCipherCtx::ProcessBlocks

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > ProcessBlocks (ReadOnlyMemRegion in, ReadWriteMemRegion out) noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleArguments	--
		if sizes of the input and output buffers are not equal
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if size of the input buffer is not divisible by the block size (see GetBlockSize())
	ara::crypto::CryptoErrc::kInvalidUsageOrder	--
		if this method is called after processing of non-aligned data (to the block-size boundary)
	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the data processing was not started by a call of the Start() method
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
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!	

⌋

[SWS_CRYPT_23615] Definition of API function ara::crypto::cryp::StreamCipherCtx::ProcessBlocks

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Scope:	`class ara::crypto::cryp::StreamCipherCtx`
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if size of the inOut buffer is not divisible by the block size (see GetBlockSize())
	ara::crypto::CryptoErrc::kInvalidUsageOrder	--
		if this method is called after processing of non-aligned data (to the block-size boundary)
	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the data processing was not started by a call of the Start() method
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!**	

⌋

[SWS_CRYPT_23616] Definition of API function ara::crypto::cryp::StreamCipherCtx::ProcessBytes

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > ProcessBytes (ReadOnlyMemRegion in, ReadWriteMemRegion out) noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if data processing was not started by a call of the Start() method
Description:	Process a non-final part of message (that is not aligned to the block-size boundary). If `(IsBytewiseMode() == 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!	

⌋

[SWS_CRYPT_23627] Definition of API function ara::crypto::cryp::StreamCipherCtx::Reset

Status: DRAFT

Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_23613] Definition of API function ara::crypto::cryp::StreamCipherCtx::Seek

Status: DRAFT

Upstream requirements: RS_CRYPTO_02304

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Seek (std::int64_t offset, bool fromBegin) 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)
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupported	--
		if the seek operation is not supported by the current mode
	ara::crypto::CryptoErrc::kProcessingNotStarted	--
		if the data processing was not started by a call of the Start() method
	ara::crypto::CryptoErrc::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)
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the offset is not aligned to the required boundary (see IsBytewiseMode())
Description:	Set the position of the next byte within the stream of the encryption/decryption gamma.	

⌋

[SWS_CRYPT_23623] Definition of API function ara::crypto::cryp::StreamCipherCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const SymmetricKey &key, CryptoTransform transform) noexcept=0;`	
Parameters (in):	key	the source key object
	transform	the transformation type "direction indicator"
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::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
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the provided transformation direction is not allowed in stream cipher algorithm context
Description:	Set (deploy) a key to the stream chiper algorithm context.	

⌋

[SWS_CRYPT_23625] Definition of API function ara::crypto::cryp::StreamCipherCtx::Start

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Start (ara::core::Optional< ReadOnlyMemRegion > iv) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by deploying a key

▽

△

	ara::crypto::CryptoErrc::kInvalidInputSize	--	
		if the size of provided IV is not supported (i.e. if it is not enough for the initialization)	
	ara::crypto::CryptoErrc::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)	
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.		

⌋

[SWS_CRYPT_23626] Definition of API function ara::crypto::cryp::StreamCipherCtx::Start

Status: DRAFT

Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::StreamCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by deploying a key
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	ara::crypto::CryptoErrc::kUnsupported	--
		if the base algorithm (or its current implementation) principally doesn't support the IV variation
	ara::crypto::CryptoErrc::kUsageViolation	--
		if this transformation type is prohibited by the "allowed usage" restrictions of the provided SecretSeed object
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.	

⌋

[SWS_CRYPT_23702] Definition of API function ara::crypto::cryp::SymmetricBlockCipherCtx::GetCryptoService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricBlockCipherCtx`	
Syntax:	`virtual CryptoService::Uptr GetCryptoService () const noexcept=0;`	
Return value:	CryptoService::Uptr	Unique smart pointer to CryptoService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get CryptoService instance.	

⌋

[SWS_CRYPT_23711] Definition of API function ara::crypto::cryp::SymmetricBlockCipherCtx::GetTransformation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricBlockCipherCtx`	
Syntax:	`virtual ara::core::Result< CryptoTransform > GetTransformation () const noexcept=0;`	
Return value:	ara::core::Result< CryptoTransform >	CryptoTransform
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if SetKey() has not been called yet.
Description:	Get the kind of transformation configured for this context: kEncrypt or kDecrypt.	

⌋

[SWS_CRYPT_23716] Definition of API function ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlock

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricBlockCipherCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > ProcessBlock (ReadOnlyMemRegion in, ReadWriteMemRegion outBuffer) const noexcept=0;`	
Parameters (in):	in	the input data block
Parameters (out):	outBuffer	the output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the provided input buffer is larger than the block size for kEncrypt
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the context requires padding and the size of the input buffer is not exactly once or twice the block size for kDecrypt
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the context does not support padding and the input buffer size is smaller than the block size
	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by calling SetKey()
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Process (encrypt / decrypt) an input block according to the configuration.	

⌋

[SWS_CRYPT_23715] Definition of API function ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlocks

Status: DRAFT
Upstream requirements: RS_CRYPTO_02302

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricBlockCipherCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > ProcessBlocks (ReadOnlyMemRegion in, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer

▽

△

Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by a key value
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if size of the input buffer is not divisible by the block size (see GetBlockSize())
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
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!**	

⌋

[SWS_CRYPT_23712] Definition of API function ara::crypto::cryp::SymmetricBlockCipherCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309, RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricBlockCipherCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	true if the transformation requires the maximum size of input data and false otherwise either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet
Description:	Indicate that the currently configured transformation accepts only complete blocks of input data. Clear the crypto context.	

⌋

[SWS_CRYPT_23710] Definition of API function ara::crypto::cryp::SymmetricBlockCipherCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricBlockCipherCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const SymmetricKey &key, CryptoTransform transform) noexcept=0;`	
Parameters (in):	key	the source key object
	transform	the transformation type "direction indicator"
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the provided key object belongs to a different CryptoProvider instance
	ara::crypto::CryptoErrc::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
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the provided transformation direction is not allowed in SymmetricBlockCipher algorithm context
Description:	Set (deploy) a key to the symmetric algorithm context.	

⌋

[SWS_CRYPT_24013] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::CalculateWrappedKeySize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual std::size_t 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:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_24002] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::GetExtensionService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	Unique smart pointer to ExtensionService
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Get ExtensionService instance.	

⌋

[SWS_CRYPT_24012] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::GetMaxTargetKeyLength

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual std::size_t GetMaxTargetKeyLength () const noexcept=0;`	
Return value:	std::size_t	maximum length of the target key in bits
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get maximum length of the target key supported by the implementation. This method can be useful for some implementations different from RFC3394 / RFC5649.	

⌋

[SWS_CRYPT_24011] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::GetTargetKeyGranularity

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`

▽

△

Syntax:	`virtual std::size_t GetTargetKeyGranularity () const noexcept=0;`	
Return value:	std::size_t	size of the block in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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).	

⌋

[SWS_CRYPT_24019] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_24018] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const SymmetricKey &key, CryptoTransform transform) noexcept=0;`	
Parameters (in):	key	the source key object
	transform	the transformation type "direction indicator"
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Errors:	ara::crypto::CryptoErrc::k IncompatibleObject	--
		if the provided key object is incompatible with this symmetric key context
	ara::crypto::CryptoErrc::k UsageViolation	--
		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
	ara::crypto::CryptoErrc::k InvalidArgument	--
		if the provided transformation direction is not allowed in Symmetric Key wrapper algorithm context
Description:	Set (deploy) a key to the symmetric key wrapper algorithm context.	

⌋

[SWS_CRYPT_24016] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02115

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual ara::core::Result< RestrictedUseObject::Uptrc > UnwrapKey (ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage) const 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidInputSize	--
		if the size of provided wrapped key is unsupported
	ara::crypto::CryptoErrc::k UninitializedContext	--
		if the context was not initialized by a key value
	ara::crypto::CryptoErrc::k UsageViolation	--
		if the kAllowKey- Importing flag of the ara::crypto::AllowedUsage Flags is not set for the Symmetric Key specified in the SetKey call.
	ara::crypto::CryptoErrc::k IncompatibleObject	--
		If the unwrapped key-material is incompatible "shorter or longer" with the target crypto algorithm specified by parameter AlgId.

▽

△

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)!

⌋

[SWS_CRYPT_24015] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapSeed

Status: DRAFT

Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual ara::core::Result< SecretSeed::Uptrc > UnwrapSeed (ReadOnlyMemRegion wrappedSeed, AlgId algId, SecretSeed::Usage allowedUsage) const noexcept=0;`	
Parameters (in):	wrappedSeed	a memory region that contains wrapped seed
	algId	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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the size of provided wrapped seed is unsupported
	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by a key value
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the kAllowKey- Importing flag of the ara::crypto::AllowedUsageFlags is not set for the Symmetric Key specified in the SetKey call.
	ara::crypto::CryptoErrc::kIncompatibleObject	--
		If the unwrapped key-material is incompatible "shorter or longer" with the target crypto algorithm specified by parameter AlgId.
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).	

⌋

[SWS_CRYPT_24014] Definition of API function ara::crypto::cryp::SymmetricKeyWrapperCtx::WrapKeyMaterial

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`	
Syntax:	`virtual ara::core::Result< std::size_t > WrapKeyMaterial (const RestrictedUseObject &key, ReadWriteMemRegion out) const noexcept=0;`	
Parameters (in):	key	a RestrictedUseObject that should be wrapped
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the key object has an unsupported length
	ara::crypto::CryptoErrc::kUninitializedContext	--
		if the context was not initialized by setting a wrapping key
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the kAllowExport flag is not set in the AllowedUsageFlags of the provided RestrictedUseObject
	ara::crypto::CryptoErrc::kUsageViolation	--
		if the kAllowKeyExporting flag of the AllowedUsageFlags is not set for the SymmetricKey specified in the SetKey call
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
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.	

⌋

[SWS_CRYPT_24102] Definition of API function ara::crypto::cryp::VerifierPublicCtx::GetSignatureService

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`	
Syntax:	`virtual SignatureService::Uptr GetSignatureService () const noexcept=0;`	
Return value:	SignatureService::Uptr	Unique smart pointer to SignatureService

▽

△

Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Extension service member class.

⌋

[SWS_CRYPT_24116] Definition of API function ara::crypto::cryp::VerifierPublicCtx::Reset

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`	
Syntax:	`virtual ara::core::Result< void > Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Clear the crypto context.	

⌋

[SWS_CRYPT_24115] Definition of API function ara::crypto::cryp::VerifierPublicCtx::SetKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02001, RS_CRYPTO_02003

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`	
Syntax:	`virtual ara::core::Result< void > SetKey (const PublicKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
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
Description:	Set (deploy) a key to the verifier public algorithm context.	

⌋

[SWS_CRYPT_24111] Definition of API function ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`	
Syntax:	`virtual ara::core::Result< bool > VerifyPrehashed (CryptoAlgId hashAlgId, ReadOnlyMemRegion hashValue, ReadOnlyMemRegion signature, ara::crypto::Serializable::FormatId fmt) const noexcept=0;`	
Parameters (in):	hashAlgId	hash function algorithm ID
	hashValue	hash function value (resulting digest without any truncations)
	signature	the signature for verification
	fmt	signature format
Return value:	ara::core::Result< bool >	true if the signature was verified successfully and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	CryptoErrc::kUnsupportedFormat	--
		if the provided parameter signature cannot be parsed according to the specified FormatId.
	CryptoErrc::kUninitializedContext	--
		if SetKey was not called before
	CryptoErrc::kInvalidArgument	--
		if the CryptoAlgId hashAlgId is incompatable with the configured signature algorithm.
	CryptoErrc::kInvalidInputSize	--
		if the size of the supplied ReadOnlyMemRegion signature or hashValue is incompatible with the configured signature algorithm.
Description:	Verify signature by digest hash function value. This is a pass-through interface to SWS_CRYPT_24112 for developer convenience, i.e. it adds additional input checks and then calls the verify() interface from SWS_CRYPT_24112.	

⌋

[SWS_CRYPT_24112] Definition of API function ara::crypto::cryp::VerifierPublicCtx::Verify

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`

▽

△

Syntax:	`virtual ara::core::Result< bool > Verify (ReadOnlyMemRegion value, ReadOnlyMemRegion signature, ara::crypto::Serializable::FormatId fmt) 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)
	fmt	the signature format
Return value:	ara::core::Result< bool >	true if the signature was verified successfully and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	CryptoErrc::k UnsupportedFormat	--
		if the provided parameter signature cannot be parsed according to the specified FormatId.
	CryptoErrc::k UninitializedContext	--
		if the context was not initialized by a key value
	CryptoErrc::kInvalidInput Size	--
		if a supplied ara::crypto::ReadOnlyMemRegion parameter size is incompatible with the configured signature algorithm
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).	

⌋

[SWS_CRYPT_24114] Definition of API function ara::crypto::cryp::VerifierPublicCtx::VerifyPrehashed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`	
Syntax:	`virtual ara::core::Result< bool > VerifyPrehashed (const HashFunctionCtx &hashFn, ReadOnlyMemRegion signature, ara::crypto::Serializable::FormatId fmt) const noexcept=0;`	
Parameters (in):	hashFn	hash function to be used for hashing
	signature	the data BLOB to be verified
	fmt	the signature format
Return value:	ara::core::Result< bool >	true if the signature was verified successfully and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Errors:	CryptoErrc::k UnsupportedFormat	--
		if the provided parameter signature cannot be parsed according to the specified FormatId.
	CryptoErrc::k UninitializedContext	--
		if SetKey was not called before
	CryptoErrc::kProcessing NotFinished	--
		if the method hashFn.Finish() was not called before this method call
	CryptoErrc::kInvalid Argument	--
		if the CryptoAlgId of hashFn is incompatible with the configured signature algorithm.
	CryptoErrc::kInvalidInput Size	--
		if the size of the supplied ReadOnlyMemRegion signature is incompatible with the configured signature algorithm.
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 and then calls the default verify() interface.	

⌋

[SWS_CRYPT_41039] Definition of API function ara::crypto::cryp::VerifierPublicCtx::AddContextData

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204, RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`	
Syntax:	`virtual ara::core::Result< void > AddContextData (ReadOnlyMemRegion context) const noexcept=0;`	
Parameters (in):	context	an optional user-supplied "context" (its support depends on concrete algorithm)
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::k Unsupported	--
		if the configured signature algorithm does not support context data.
Description:	Add context data as raw bytes. Whether or not this data is needed depends on the signature algorithm.	

⌋

[SWS_CRYPT_24101] Definition of API type ara::crypto::cryp::VerifierPublicCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"
Scope:	`class ara::crypto::cryp::VerifierPublicCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<VerifierPublicCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_20101] Definition of API type ara::crypto::cryp::AuthCipherCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02207

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"
Scope:	`class ara::crypto::cryp::AuthCipherCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<AuthCipherCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_24802] Definition of API type ara::crypto::cryp::RestrictedUseObject::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02403

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/restricted_use_object.h"
Scope:	`class ara::crypto::cryp::RestrictedUseObject`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const RestrictedUseObject>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_29031] Definition of API type ara::crypto::cryp::BlockService::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/block_service.h"
Scope:	`class ara::crypto::cryp::BlockService`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<BlockService>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_20402] Definition of API type ara::crypto::cryp::CryptoContext::AlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Scope:	`class ara::crypto::cryp::CryptoContext`
Symbol:	AlgId
Syntax:	`using AlgId = CryptoAlgId;`
Description:	Type definition of vendor specific binary Crypto Primitive ID.

⌋

[SWS_CRYPT_20504] Definition of API class ara::crypto::cryp::CryptoObject::COIdentifier

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	struct
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Symbol:	COIdentifier
Syntax:	`struct COIdentifier {...};`
Description:	Unique identifier of this CryptoObject.

⌋

[SWS_CRYPT_20502] Definition of API type ara::crypto::cryp::CryptoObject::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const CryptoObject>;`
Description:	Unique smart pointer of the constant interface.

⌋

[SWS_CRYPT_20501] Definition of API type ara::crypto::cryp::CryptoObject::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`class ara::crypto::cryp::CryptoObject`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<CryptoObject>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_20641] Definition of API type ara::crypto::cryp::CryptoPrimitiveId::AlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Symbol:	AlgId
Syntax:	`using AlgId = CryptoAlgId;`
Description:	Type definition of vendor specific binary Crypto Primitive ID.

⌋

[SWS_CRYPT_20644] Definition of API type ara::crypto::cryp::CryptoPrimitiveId::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const CryptoPrimitiveId>;`
Description:	type definition pointer to const

⌋

[SWS_CRYPT_20643] Definition of API type ara::crypto::cryp::CryptoPrimitiveId::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Scope:	`class ara::crypto::cryp::CryptoPrimitiveId`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<CryptoPrimitiveId>;`
Description:	type definition pointer

⌋

[SWS_CRYPT_20703] Definition of API type ara::crypto::cryp::CryptoProvider::AlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005, RS_CRYPTO_02006

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Symbol:	AlgId
Syntax:	`using AlgId = CryptoPrimitiveId::AlgId;`
Description:	A short alias for Algorithm ID type definition.

⌋

[SWS_CRYPT_20701] Definition of API type ara::crypto::cryp::CryptoProvider::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Scope:	`class ara::crypto::cryp::CryptoProvider`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<CryptoProvider>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_29024] Definition of API type ara::crypto::cryp::CryptoService::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/crypto_service.h"
Scope:	`class ara::crypto::cryp::CryptoService`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<CryptoService>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_20801] Definition of API type ara::crypto::cryp::DecryptorPrivateCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"
Scope:	`class ara::crypto::cryp::DecryptorPrivateCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<DecryptorPrivateCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_29011] Definition of API type ara::crypto::cryp::DigestService::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/digest_service.h"
Scope:	`class ara::crypto::cryp::DigestService`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<DigestService>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_21001] Definition of API type ara::crypto::cryp::EncryptorPublicCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"
Scope:	`class ara::crypto::cryp::EncryptorPublicCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<EncryptorPublicCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_29042] Definition of API type ara::crypto::cryp::ExtensionService::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/extension_service.h"
Scope:	`class ara::crypto::cryp::ExtensionService`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<ExtensionService>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_21101] Definition of API type ara::crypto::cryp::HashFunctionCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02205

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"
Scope:	`class ara::crypto::cryp::HashFunctionCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<HashFunctionCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_21301] Definition of API type ara::crypto::cryp::KeyAgreementPrivateCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"
Scope:	`class ara::crypto::cryp::KeyAgreementPrivateCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeyAgreementPrivateCtx>;`
Description:	Unique smart pointer of this interface.

⌋

[SWS_CRYPT_21401] Definition of API type ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02104

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"
Scope:	`class ara::crypto::cryp::KeyDecapsulatorPrivateCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeyDecapsulatorPrivateCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_21501] Definition of API type ara::crypto::cryp::KeyDerivationFunctionCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02103

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"
Scope:	`class ara::crypto::cryp::KeyDerivationFunctionCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeyDerivationFunctionCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_21801] Definition of API type ara::crypto::cryp::KeyEncapsulatorPublicCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02209

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"
Scope:	`class ara::crypto::cryp::KeyEncapsulatorPublicCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeyEncapsulatorPublicCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_22101] Definition of API type ara::crypto::cryp::MessageAuthnCodeCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02203

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"
Scope:	`class ara::crypto::cryp::MessageAuthnCodeCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<MessageAuthnCodeCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_22201] Definition of API type ara::crypto::cryp::MsgRecoveryPublicCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"
Scope:	`class ara::crypto::cryp::MsgRecoveryPublicCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<MsgRecoveryPublicCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_22501] Definition of API type ara::crypto::cryp::PrivateKey::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/private_key.h"
Scope:	`class ara::crypto::cryp::PrivateKey`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const PrivateKey>;`
Description:	Unique smart pointer of a constant interface instance.

⌋

[SWS_CRYPT_22701] Definition of API type ara::crypto::cryp::PublicKey::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"
Scope:	`class ara::crypto::cryp::PublicKey`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const PublicKey>;`
Description:	Unique smart pointer of a constant interface instance.

⌋

[SWS_CRYPT_22901] Definition of API type ara::crypto::cryp::RandomGeneratorCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02206

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"
Scope:	`class ara::crypto::cryp::RandomGeneratorCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<RandomGeneratorCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_24801] Definition of API type ara::crypto::cryp::RestrictedUseObject::Usage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/restricted_use_object.h"
Scope:	`class ara::crypto::cryp::RestrictedUseObject`
Symbol:	Usage
Syntax:	`using Usage = AllowedUsageFlags;`
Description:	Alias to the container type for bit-flags of allowed usages of the object.

⌋

[SWS_CRYPT_23001] Definition of API type ara::crypto::cryp::SecretSeed::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Scope:	`class ara::crypto::cryp::SecretSeed`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const SecretSeed>;`
Description:	Unique smart pointer of a constant interface instance.

⌋

[SWS_CRYPT_23002] Definition of API type ara::crypto::cryp::SecretSeed::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Scope:	`class ara::crypto::cryp::SecretSeed`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<SecretSeed>;`
Description:	Unique smart pointer of a volatile interface instance.

⌋

[SWS_CRYPT_23201] Definition of API type ara::crypto::cryp::SigEncodePrivateCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204, RS_CRYPTO_02202

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Scope:	`class ara::crypto::cryp::SigEncodePrivateCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<SigEncodePrivateCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_29001] Definition of API type ara::crypto::cryp::SignatureService::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/signature_service.h"
Scope:	`class ara::crypto::cryp::SignatureService`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<SignatureService>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_23501] Definition of API type ara::crypto::cryp::SignerPrivateCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02204

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"
Scope:	`class ara::crypto::cryp::SignerPrivateCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<SignerPrivateCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_23601] Definition of API type ara::crypto::cryp::StreamCipherCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Scope:	`class ara::crypto::cryp::StreamCipherCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<StreamCipherCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_23701] Definition of API type ara::crypto::cryp::SymmetricBlockCipherCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"
Scope:	`class ara::crypto::cryp::SymmetricBlockCipherCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<SymmetricBlockCipherCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_23801] Definition of API type ara::crypto::cryp::SymmetricKey::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/cryobj/symmetric_key.h"
Scope:	`class ara::crypto::cryp::SymmetricKey`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const SymmetricKey>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_24001] Definition of API type ara::crypto::cryp::SymmetricKeyWrapperCtx::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	type alias
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"
Scope:	`class ara::crypto::cryp::SymmetricKeyWrapperCtx`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<SymmetricKeyWrapperCtx>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_20506] Definition of API variable ara::crypto::cryp::CryptoObject::COIdentifier::mCOType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	variable
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`struct ara::crypto::cryp::CryptoObject::COIdentifier`
Symbol:	mCOType
Type:	`CryptoObjectType`
Syntax:	`CryptoObjectType mCOType;`
Description:	type of objext

⌋

[SWS_CRYPT_20507] Definition of API variable ara::crypto::cryp::CryptoObject::COIdentifier::mCouid

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	variable
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Scope:	`struct ara::crypto::cryp::CryptoObject::COIdentifier`
Symbol:	mCouid
Type:	`CryptoObjectUid`
Syntax:	`CryptoObjectUid mCouid;`
Description:	object identifier

⌋

[SWS_CRYPT_22503] Definition of API variable ara::crypto::cryp::PrivateKey::kObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/cryp/cryobj/private_key.h"
Scope:	`class ara::crypto::cryp::PrivateKey`
Symbol:	kObjectType
Type:	`const CryptoObjectType`
Syntax:	`static const CryptoObjectType kObjectType {CryptoObjectType::kPrivateKey};`
Description:	Static mapping of this interface to specific value of `CryptoObjectType` enumeration.

⌋

[SWS_CRYPT_22702] Definition of API variable ara::crypto::cryp::PublicKey::kObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02202

⌈

Kind:	variable
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"
Scope:	`class ara::crypto::cryp::PublicKey`
Symbol:	kObjectType
Type:	`const CryptoObjectType`

▽

△

Syntax:	`static const CryptoObjectType kObjectType {CryptoObjectType::kPublicKey};`
Description:	const object type

⌋

[SWS_CRYPT_23003] Definition of API variable ara::crypto::cryp::SecretSeed::kObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02007

⌈

Kind:	variable
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Scope:	`class ara::crypto::cryp::SecretSeed`
Symbol:	kObjectType
Type:	`const CryptoObjectType`
Syntax:	`static const CryptoObjectType kObjectType {CryptoObjectType::kSecretSeed};`
Description:	Static mapping of this interface to specific value of `CryptoObjectType` enumeration.

⌋

[SWS_CRYPT_23802] Definition of API variable ara::crypto::cryp::SymmetricKey::kObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02201

⌈

Kind:	variable
Header file:	#include "ara/crypto/cryp/cryobj/symmetric_key.h"
Scope:	`class ara::crypto::cryp::SymmetricKey`
Symbol:	kObjectType
Type:	`const CryptoObjectType`
Syntax:	`static const CryptoObjectType kObjectType {CryptoObjectType::kSymmetricKey};`
Description:	const object type

⌋

[SWS_CRYPT_10101] Definition of API variable ara::crypto::CryptoObjectUid::mGeneratorUid

Status: DRAFT

Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Scope:	`struct ara::crypto::CryptoObjectUid`
Symbol:	mGeneratorUid
Type:	`Uuid`
Syntax:	`Uuid mGeneratorUid;`
Description:	UUID of a generator that has produced this COUID. This UUID can be associated with HSM, physical host/ECU or VM.

⌋

8.2 C++ language binding Key Storage Provider

[SWS_CRYPT_30400] Definition of API class ara::crypto::keys::KeySlot

Status: DRAFT

Upstream requirements: RS_CRYPTO_02405

⌈

Kind:	class
Header file:	#include "ara/crypto/keys/keyslot.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::keys`
Symbol:	KeySlot
Syntax:	`class KeySlot {...};`
Description:	Key slot port-prototype interface. This class enables access to a physicl key-slot.

⌋

[SWS_CRYPT_30100] Definition of API class ara::crypto::keys::KeyStorageProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109, RS_CRYPTO_02305, RS_CRYPTO_02401

⌈

Kind:	class
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::keys`
Symbol:	KeyStorageProvider
Syntax:	`class KeyStorageProvider {...};`
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.

⌋

[SWS_CRYPT_30200] Definition of API class ara::crypto::keys::UpdatesObserver

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	class
Header file:	#include "ara/crypto/keys/updates_observer.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::keys`
Symbol:	UpdatesObserver
Syntax:	`class UpdatesObserver {...};`
Description:	Definition of an "updates observer" interface. 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.

⌋

[SWS_CRYPT_30405] Definition of API function ara::crypto::keys::KeySlot::Clear

Status: DRAFT
Upstream requirements: RS_CRYPTO_02009

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`

▽

△

Syntax:	`virtual ara::core::Result< void > Clear () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnreservedResource	--
		if the target slot is not opened writeable.
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()`.	

⌋

[SWS_CRYPT_30510] Definition of API function ara::crypto::keys::KeySlotContentProps::KeySlotContentProps

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Scope:	`struct ara::crypto::keys::KeySlotContentProps`
Syntax:	`KeySlotContentProps ()=default;`
Thread Safety:	implementation defined
Description:	Constructor.

⌋

[SWS_CRYPT_30401] Definition of API function ara::crypto::keys::KeySlot::~KeySlot

Status: DRAFT
Upstream requirements: RS_CRYPTO_02405

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`
Syntax:	`virtual ~KeySlot () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_30408] Definition of API function ara::crypto::keys::KeySlot::GetContentProps

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Scope:	`class ara::crypto::keys::KeySlot`	
Syntax:	`virtual ara::core::Result< KeySlotContentProps > GetContentProps () const noexcept=0;`	
Return value:	ara::core::Result< KeySlotContentProps >	actual properties of a content in the key slot.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kEmptyContainer	--
		if the slot is empty
	ara::crypto::CryptoErrc::kAccessViolation	--
		if this method is called by an Actor, which has no any("Owner" or "User") access rights to the key slot
Description:	Get an actual properties of a content in the key slot.	

⌋

[SWS_CRYPT_30403] Definition of API function ara::crypto::keys::KeySlot::MyProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Scope:	`class ara::crypto::keys::KeySlot`	
Syntax:	`virtual ara::core::Result< cryp::CryptoProvider::Uptr > MyProvider () const noexcept=0;`	
Return value:	ara::core::Result< cryp::CryptoProvider::Uptr >	a unique_pointer to the CryptoProvider to be used with this KeySlot
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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. .	

⌋

[SWS_CRYPT_30407] Definition of API function ara::crypto::keys::KeySlot::GetPrototypedProps

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Scope:	`class ara::crypto::keys::KeySlot`	
Syntax:	`virtual ara::core::Result< KeySlotPrototypeProps > GetPrototypedProps () const noexcept=0;`	
Return value:	ara::core::Result< KeySlotPrototypeProps >	the prototype properties of the key slot.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the prototyped properties of the key slot.	

⌋

[SWS_CRYPT_30404] Definition of API function ara::crypto::keys::KeySlot::IsEmpty

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Scope:	`class ara::crypto::keys::KeySlot`	
Syntax:	`virtual bool IsEmpty () const noexcept=0;`	
Return value:	bool	true if the slot is empty or false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check the slot for emptiness.	

⌋

[SWS_CRYPT_30409] Definition of API function ara::crypto::keys::KeySlot::Open

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`

▽

△

Syntax:	`virtual ara::core::Result< IOInterface::Uptr > Open (ara::core::Optional< bool > writeable) const noexcept=0;`	
Parameters (in):	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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::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
	ara::crypto::CryptoErrc::kModifiedResource	--
		if the specified slot has been modified after the KeySlot has been opened
Description:	This interface shall open the KeySlot and return an IOInterface pointing to the KeySlot content.	

⌋

[SWS_CRYPT_41031] Definition of API function ara::crypto::keys::KeySlot::SubscribeForUpdates

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`
Syntax:	`virtual void SubscribeForUpdates () noexcept=0;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Requests the KeyStorageProvider to monitor the KeySlot for changes. Notifications to be sent to the application through a registered `UpdatesObserver`. .

⌋

[SWS_CRYPT_41032] Definition of API function ara::crypto::keys::KeySlot::UnsubscribeFromUpdates

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`

▽

△

Syntax:	`virtual void UnsubscribeFromUpdates () noexcept=0;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	KeyStorageProvider stops monitoring the key slot for changes for the calling application.The method ensures that the application will not receive notifications about changes to this keyslot. .

⌋

[SWS_CRYPT_30301] Definition of API function ara::crypto::keys::KeySlotPrototypeProps::KeySlotPrototypeProps

Status: DRAFT

Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Syntax:	`KeySlotPrototypeProps ()=default;`
Thread Safety:	implementation defined
Description:	Constructor.

⌋

[SWS_CRYPT_30406] Definition of API function ara::crypto::keys::KeySlot::SaveCopy

Status: DRAFT

Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Scope:	`class ara::crypto::keys::KeySlot`	
Syntax:	`virtual ara::core::Result< void > SaveCopy (const IOInterface &container) noexcept=0;`	
Parameters (in):	container	the source IOInterface
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompatibleObject	--
		if the source object has property "session" or if the source IOInterface references a KeySlot from a different CryptoProvider
	ara::crypto::CryptoErrc::kEmptyContainer	--
		if the source IOInterface is empty

▽

△

	ara::crypto::CryptoErrc::kContentRestrictions	--	
		if the source object doesn't satisfy the slot restrictions (including version control)	
	ara::crypto::CryptoErrc::kUnreservedResource	--	
		if the target slot is not opened writeable.	
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()`.		

⌋

[SWS_CRYPT_30220] Definition of API function ara::crypto::keys::KeySlot::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`
Syntax:	`KeySlot & operator= (const KeySlot &other)=delete;`
Description:	Copy-assign another KeySlot to this instance.

⌋

[SWS_CRYPT_30221] Definition of API function ara::crypto::keys::KeySlot::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`
Syntax:	`KeySlot & operator= (KeySlot &&other)=delete;`
Description:	Move-assign another KeySlot to this instance.

⌋

[SWS_CRYPT_41009] Definition of API function ara::crypto::keys::KeySlot::KeySlot

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`
Syntax:	`KeySlot (const KeySlot &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41010] Definition of API function ara::crypto::keys::KeySlot::KeySlot

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`
Syntax:	`KeySlot (KeySlot &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_30123] Definition of API function ara::crypto::keys::KeyStorageProvider::BeginTransaction

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Scope:	`class ara::crypto::keys::KeyStorageProvider`	
Syntax:	`virtual ara::core::Result< TransactionId > 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:	exception safe	
Thread Safety:	thread-safe	

▽

△

Errors:	ara::crypto::CryptoErrc::kUnreservedResource	--
		if targetSlots list has a slot that has not been configured with the reserveSpareSlot parameter in the manifest
	ara::crypto::CryptoErrc::kBusyResource	--
		if targetSlots list has key slots that are already involved to another pending transaction or opened in writing mode
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().	

⌋

[SWS_CRYPT_30124] Definition of API function ara::crypto::keys::KeyStorageProvider::CommitTransaction

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Scope:	`class ara::crypto::keys::KeyStorageProvider`	
Syntax:	`virtual ara::core::Result< void > CommitTransaction (TransactionId id) noexcept=0;`	
Parameters (in):	id	an ID of a transaction that should be commited
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back)
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.	

⌋

[SWS_CRYPT_30110] Definition of API function ara::crypto::keys::KeyStorageProvider::~KeyStorageProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Scope:	`class ara::crypto::keys::KeyStorageProvider`

▽

△

Syntax:	`virtual ~KeyStorageProvider () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_30115] Definition of API function ara::crypto::keys::KeyStorageProvider::LoadKeySlot

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Scope:	`class ara::crypto::keys::KeyStorageProvider`	
Syntax:	`virtual ara::core::Result< KeySlot::Uptr > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnreservedResource	--
		if the InstanceSpecifier is incorrect (the slot is not allocated)
Violations:	`ProcessMappingViolation`	In case InstanceSpecifier does not point to a PortPrototype typed by a CryptoKeySlotInterface modeled for the current process.
Description:	Load a key slot. The functions loads the information associated with a KeySlot into a KeySlot object.	

⌋

[SWS_CRYPT_30130] Definition of API function ara::crypto::keys::KeyStorageProvider::RegisterObserver

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Scope:	`class ara::crypto::keys::KeyStorageProvider`	
Syntax:	`virtual ara::core::Result< void > RegisterObserver (UpdatesObserver::Uptr observer) noexcept=0;`	
Parameters (in):	observer	pointer to a client-supplied UpdatesObserver instance that should be registered

▽

△

Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the parameter observer equals the nullptr
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.	

⌋

[SWS_CRYPT_41030] Definition of API function ara::crypto::keys::KeyStorageProvider::UnregisterObserver
Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Scope:	`class ara::crypto::keys::KeyStorageProvider`
Syntax:	`virtual void UnregisterObserver () noexcept=0;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Unregister consumer `UpdatesObserver` that was registered by the calling process. Note: Nothing will occur if there is no registered observer.

⌋

[SWS_CRYPT_30125] Definition of API function ara::crypto::keys::KeyStorageProvider::RollbackTransaction
Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Scope:	`class ara::crypto::keys::KeyStorageProvider`	
Syntax:	`virtual ara::core::Result< void > RollbackTransaction (TransactionId id) noexcept=0;`	
Parameters (in):	id	an ID of a transaction that should be rolled back
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--

▽

△

		if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back)
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.	

⌋

[SWS_CRYPT_30222] Definition of API function ara::crypto::keys::KeyStorageProvider::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Scope:	`class ara::crypto::keys::KeyStorageProvider`
Syntax:	`KeyStorageProvider & operator= (const KeyStorageProvider &other)=delete;`
Description:	Copy-assign another KeyStorageProvider to this instance.

⌋

[SWS_CRYPT_30223] Definition of API function ara::crypto::keys::KeyStorageProvider::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Scope:	`class ara::crypto::keys::KeyStorageProvider`
Syntax:	`KeyStorageProvider & operator= (KeyStorageProvider &&other)=delete;`
Description:	Move-assign another KeyStorageProvider to this instance.

⌋

[SWS_CRYPT_41011] Definition of API function ara::crypto::keys::KeyStorageProvider::KeyStorageProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Scope:	`class ara::crypto::keys::KeyStorageProvider`
Syntax:	`KeyStorageProvider (const KeyStorageProvider &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41012] Definition of API function ara::crypto::keys::KeyStorageProvider::KeyStorageProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Scope:	`class ara::crypto::keys::KeyStorageProvider`
Syntax:	`KeyStorageProvider (KeyStorageProvider &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_30350] Definition of API function ara::crypto::keys::operator==

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"	
Scope:	`namespace ara::crypto::keys`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "equal" for `KeySlotPrototypeProps` operands.	

⌋

[SWS_CRYPT_30351] Definition of API function ara::crypto::keys::operator!=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"	
Scope:	`namespace ara::crypto::keys`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "not equal" for `KeySlotPrototypeProps` operands.	

⌋

[SWS_CRYPT_30550] Definition of API function ara::crypto::keys::operator==

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"	
Scope:	`namespace ara::crypto::keys`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "equal" for `KeySlotContentProps` operands.	

⌋

[SWS_CRYPT_30551] Definition of API function ara::crypto::keys::operator!=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	function	
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"	
Scope:	`namespace ara::crypto::keys`	
Syntax:	`constexpr bool 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 at least one member of lhs has a value not equal to correspondent member of rhs, and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "not equal" for `KeySlotContentProps` operands.	

⌋

[SWS_CRYPT_30210] Definition of API function ara::crypto::keys::UpdatesObserver::~UpdatesObserver

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/updates_observer.h"
Scope:	`class ara::crypto::keys::UpdatesObserver`
Syntax:	`virtual ~UpdatesObserver () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_30211] Definition of API function ara::crypto::keys::UpdatesObserver::OnUpdate

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/updates_observer.h"
Scope:	`class ara::crypto::keys::UpdatesObserver`

▽

△

Syntax:	`virtual void OnUpdate (ara::core::Vector< ara::core::InstanceSpecifier > &updatedSlotsSpecifiers) noexcept=0;`	
Parameters (in):	updatedSlotsSpecifiers	List of monitored slots that were updated
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	This method is implemented by the user application and used by FC Crypto to notify the application that key-slots have been updated to which the user has subscribed. Note: each slot is presented in the provided list only one time!	

⌋

[SWS_CRYPT_30224] Definition of API function ara::crypto::keys::UpdatesObserver::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/updates_observer.h"
Scope:	`class ara::crypto::keys::UpdatesObserver`
Syntax:	`UpdatesObserver & operator= (const UpdatesObserver &other)=delete;`
Description:	Copy-assign another UpdatesObserver to this instance.

⌋

[SWS_CRYPT_30225] Definition of API function ara::crypto::keys::UpdatesObserver::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/updates_observer.h"
Scope:	`class ara::crypto::keys::UpdatesObserver`
Syntax:	`UpdatesObserver & operator= (UpdatesObserver &&other)=delete;`
Description:	Move-assign another UpdatesObserver to this instance.

⌋

[SWS_CRYPT_41015] Definition of API function ara::crypto::keys::UpdatesObserver::UpdatesObserver

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/updates_observer.h"
Scope:	`class ara::crypto::keys::UpdatesObserver`
Syntax:	`UpdatesObserver (const UpdatesObserver &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41016] Definition of API function ara::crypto::keys::UpdatesObserver::UpdatesObserver

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/keys/updates_observer.h"
Scope:	`class ara::crypto::keys::UpdatesObserver`
Syntax:	`UpdatesObserver (UpdatesObserver &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_30500] Definition of API class ara::crypto::keys::KeySlotContentProps

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005, RS_CRYPTO_02111

⌈

Kind:	struct
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::keys`
Symbol:	KeySlotContentProps
Syntax:	`struct KeySlotContentProps {...};`
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".

⌋

[SWS_CRYPT_30511] Definition of API type ara::crypto::keys::KeySlotContentProps::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	type alias
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Scope:	`struct ara::crypto::keys::KeySlotContentProps`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeySlotContentProps>;`
Description:	unique smart pointer of interface

⌋

[SWS_CRYPT_30300] Definition of API class ara::crypto::keys::KeySlotPrototypeProps

Status: DRAFT
Upstream requirements: RS_CRYPTO_02009, RS_CRYPTO_02110, RS_CRYPTO_02116

⌈

Kind:	struct
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::keys`
Symbol:	KeySlotPrototypeProps
Syntax:	`struct KeySlotPrototypeProps {...};`
Description:	Prototyped Properties of a Key Slot.

⌋

[SWS_CRYPT_30302] Definition of API type ara::crypto::keys::KeySlotPrototypeProps::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	type alias
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeySlotPrototypeProps>;`
Description:	unique smart pointer of interface

⌋

[SWS_CRYPT_30402] Definition of API type ara::crypto::keys::KeySlot::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02405

⌈

Kind:	type alias
Header file:	#include "ara/crypto/keys/keyslot.h"
Scope:	`class ara::crypto::keys::KeySlot`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeySlot>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_30101] Definition of API type ara::crypto::keys::KeyStorageProvider::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	type alias
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Scope:	`class ara::crypto::keys::KeyStorageProvider`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<KeyStorageProvider>;`
Description:	unique smart pointer of interface

⌋

[SWS_CRYPT_30010] Definition of API type ara::crypto::keys::TransactionId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	type alias
Header file:	#include "ara/crypto/keys/elementary_types.h"
Scope:	`namespace ara::crypto::keys`
Symbol:	TransactionId
Syntax:	`using TransactionId = std::uint64_t;`
Description:	Definition of a transaction identifier type. The zero value should be reserved for especial cases.

⌋

[SWS_CRYPT_30011] Definition of API type ara::crypto::keys::TransactionScope

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	type alias
Header file:	#include "ara/crypto/keys/elementary_types.h"
Scope:	`namespace ara::crypto::keys`
Symbol:	TransactionScope
Syntax:	`using TransactionScope = ara::core::Vector<KeySlot>;`
Description:	Definition of a "transaction scope" type. The "transaction scope" defines a list of key slots that are target for update in a transaction.

⌋

[SWS_CRYPT_30201] Definition of API type ara::crypto::keys::UpdatesObserver::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	type alias
Header file:	#include "ara/crypto/keys/updates_observer.h"
Scope:	`class ara::crypto::keys::UpdatesObserver`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<UpdatesObserver>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_30503] Definition of API variable ara::crypto::keys::KeySlotContentProps::mAlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Scope:	`struct ara::crypto::keys::KeySlotContentProps`
Symbol:	mAlgId
Type:	`CryptoAlgId`
Syntax:	`CryptoAlgId mAlgId;`
Description:	Cryptoalgorithm of actual object stored to the slot.

⌋

[SWS_CRYPT_30505] Definition of API variable ara::crypto::keys::KeySlotContentProps::mObjectSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Scope:	`struct ara::crypto::keys::KeySlotContentProps`
Symbol:	mObjectSize
Type:	`std::size_t`
Syntax:	`std::size_t mObjectSize;`
Description:	Actual size of an object currently stored to the slot.

⌋

[SWS_CRYPT_30508] Definition of API variable ara::crypto::keys::KeySlotContentProps::mObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Scope:	`struct ara::crypto::keys::KeySlotContentProps`
Symbol:	mObjectType
Type:	`CryptoObjectType`
Syntax:	`CryptoObjectType mObjectType;`
Description:	Actual type of an object stored to the slot.

⌋

[SWS_CRYPT_30501] Definition of API variable ara::crypto::keys::KeySlotContentProps::mObjectUid

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Scope:	`struct ara::crypto::keys::KeySlotContentProps`
Symbol:	mObjectUid
Type:	`CryptoObjectUid`
Syntax:	`CryptoObjectUid mObjectUid;`

▽

△

Description:	UID of a Crypto Object stored to the slot.

⌋

[SWS_CRYPT_30506] Definition of API variable ara::crypto::keys::KeySlotContentProps::mContentAllowedUsage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Scope:	`struct ara::crypto::keys::KeySlotContentProps`
Symbol:	mContentAllowedUsage
Type:	`AllowedUsageFlags`
Syntax:	`AllowedUsageFlags mContentAllowedUsage;`
Description:	Actual usage restriction flags of an object stored to the slot for the current "Actor".

⌋

[SWS_CRYPT_30306] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mAlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mAlgId
Type:	`CryptoAlgId`
Syntax:	`CryptoAlgId mAlgId;`
Description:	Cryptoalgorithm restriction The algorithm can be specified partially: family & length, mode, padding.

⌋

[SWS_CRYPT_30309] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mAllocateSpareSlot

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mAllocateSpareSlot
Type:	`bool`
Syntax:	`bool mAllocateSpareSlot;`
Description:	Indicates whether FC Crypto shall allocate sufficient storage space for a shadow copy of this KeySlot.

⌋

[SWS_CRYPT_30310] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mAllowContentTypeChange

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mAllowContentTypeChange
Type:	`bool`
Syntax:	`bool mAllowContentTypeChange;`
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 KeySlotPrototypeProps must be a) valid and b) cannot be changed (i.e. only objects of mObjectType may be stored in this key-slot).

⌋

[SWS_CRYPT_30313] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mContentAllowedUsage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`

▽

△

Symbol:	mContentAllowedUsage
Type:	`AllowedUsageFlags`
Syntax:	`AllowedUsageFlags mContentAllowedUsage;`
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 kAllowPrototyped Only. 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

⌋

[SWS_CRYPT_30312] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mExportAllowed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mExportAllowed
Type:	`bool`
Syntax:	`bool mExportAllowed;`
Description:	Indicates whether the key-slot content may be exported.

⌋

[SWS_CRYPT_30311] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mMaxUpdateAllowed

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mMaxUpdateAllowed
Type:	`std::int32_t`
Syntax:	`std::int32_t mMaxUpdateAllowed;`

▽

△

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

⌋

[SWS_CRYPT_30305] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mSlotType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mSlotType
Type:	`KeySlotType`
Syntax:	`KeySlotType mSlotType;`
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.

⌋

[SWS_CRYPT_30307] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mSlotCapacity

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mSlotCapacity
Type:	`std::size_t`
Syntax:	`std::size_t mSlotCapacity;`
Description:	Capacity of the slot in bytes.

⌋

[SWS_CRYPT_30308] Definition of API variable ara::crypto::keys::KeySlotPrototypeProps::mObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	variable
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Scope:	`struct ara::crypto::keys::KeySlotPrototypeProps`
Symbol:	mObjectType
Type:	`CryptoObjectType`
Syntax:	`CryptoObjectType mObjectType;`
Description:	Restriction of an object type that can be stored the slot. If this field contains `CryptoObjectType::kUnknown` then without restriction of the type.

⌋

8.3 C++ language binding X509 Certificate Management Provider

[SWS_CRYPT_40100] Definition of API class ara::crypto::x509::BasicCertInfo

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	BasicCertInfo
Base class:	X509Object
Syntax:	`class BasicCertInfo : public X509Object {...};`
Description:	Basic Certificate Information interface.

⌋

[SWS_CRYPT_40200] Definition of API class ara::crypto::x509::Certificate
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/certificate.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	Certificate
Base class:	BasicCertInfo
Syntax:	`class Certificate : public BasicCertInfo {...};`
Description:	X.509 Certificate interface.

⌋

[SWS_CRYPT_40300] Definition of API class ara::crypto::x509::CertSignRequest
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	CertSignRequest
Base class:	BasicCertInfo
Syntax:	`class CertSignRequest : public BasicCertInfo {...};`
Description:	Certificate Signing Request (CSR) object interface This interface is dedicated for complete parsing of the request content.

⌋

[SWS_CRYPT_40700] Definition of API class ara::crypto::x509::OcspRequest
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	OcspRequest
Base class:	X509Object
Syntax:	`class OcspRequest : public X509Object {...};`

▽

△

Description:	On-line Certificate Status Protocol Request.

⌋

[SWS_CRYPT_40800] Definition of API class ara::crypto::x509::OcspResponse

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	OcspResponse
Base class:	X509Object
Syntax:	`class OcspResponse : public X509Object {...};`
Description:	On-line Certificate Status Protocol Response.

⌋

[SWS_CRYPT_24400] Definition of API class ara::crypto::x509::X509PublicKeyInfo

Status: DRAFT
Upstream requirements: RS_CRYPTO_02307

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	X509PublicKeyInfo
Base class:	ara::crypto::Serializable
Syntax:	`class X509PublicKeyInfo : public ara::crypto::Serializable {...};`
Description:	X.509 Public Key Information interface.

⌋

[SWS_CRYPT_40400] Definition of API class ara::crypto::x509::X509DN

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/x509_dn.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	X509DN
Base class:	X509Object
Syntax:	`class X509DN : public X509Object {...};`
Description:	Interface of X.509 Distinguished Name (DN).

⌋

[SWS_CRYPT_40500] Definition of API class ara::crypto::x509::X509Extensions

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/x509_extensions.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	X509Extensions
Base class:	X509Object
Syntax:	`class X509Extensions :  public X509Object {...};`
Description:	Interface of X.509 Extensions.

⌋

[SWS_CRYPT_40900] Definition of API class ara::crypto::x509::X509Object

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/x509_object.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	X509Object
Base class:	ara::crypto::Serializable
Syntax:	`class X509Object :  public ara::crypto::Serializable {...};`

▽

△

Description:	Common interface of all objects created by X.509 Provider.

⌋

[SWS_CRYPT_40600] Definition of API class ara::crypto::x509::X509Provider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/x509_provider.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	X509Provider
Syntax:	`class X509Provider {...};`
Description:	X.509 Provider interface.

⌋

[SWS_CRYPT_40932] Definition of API class ara::crypto::x509::X509CustomExtensionsParser

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	class
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto::x509`
Symbol:	X509CustomExtensionsParser
Syntax:	`class X509CustomExtensionsParser {...};`
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.

⌋

[SWS_CRYPT_24414] Definition of API function ara::crypto::x509::X509PublicKeyInfo::GetPublicKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02108, RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Scope:	`class ara::crypto::x509::X509PublicKeyInfo`	
Syntax:	`virtual ara::core::Result< ara::crypto::cryp::PublicKey::Uptrc > 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:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_24412] Definition of API function ara::crypto::x509::X509PublicKeyInfo::GetRequiredHashAlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Scope:	`class ara::crypto::x509::X509PublicKeyInfo`	
Syntax:	`virtual CryptoAlgId GetRequiredHashAlgId () const noexcept=0;`	
Return value:	CryptoAlgId	required hash algorithm ID
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get an ID of hash algorithm required by current signature algorithm.	

⌋

[SWS_CRYPT_24411] Definition of API function ara::crypto::x509::X509PublicKeyInfo::GetRequiredHashSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Scope:	`class ara::crypto::x509::X509PublicKeyInfo`	
Syntax:	`virtual std::size_t GetRequiredHashSize () const noexcept=0;`	
Return value:	std::size_t	required hash size in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the hash size required by current signature algorithm.	

⌋

[SWS_CRYPT_24413] Definition of API function ara::crypto::x509::X509PublicKeyInfo::GetSignatureSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02309

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Scope:	`class ara::crypto::x509::X509PublicKeyInfo`	
Syntax:	`virtual std::size_t GetSignatureSize () const noexcept=0;`	
Return value:	std::size_t	size of the signature value in bytes
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get size of the signature value produced and required by the current algorithm.	

⌋

[SWS_CRYPT_24410] Definition of API function ara::crypto::x509::X509PublicKeyInfo::GetAlgorithmId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02307

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"
Scope:	`class ara::crypto::x509::X509PublicKeyInfo`
Syntax:	`virtual ara::crypto::cryp::CryptoPrimitiveId::Uptrc GetAlgorithmId ()=0;`

▽

△

Return value:	ara::crypto::cryp::Crypto PrimitiveId::Uptrc	Unique smart pointer to constant CryptoPrimitiveId.
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Get the CryptoPrimitiveId instance of this class.	

⌋

[SWS_CRYPT_24415] Definition of API function ara::crypto::x509::X509PublicKeyInfo::IsSameKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Scope:	`class ara::crypto::x509::X509PublicKeyInfo`	
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Verify the sameness of the provided and kept public keys. This method compare the public key values only.	

⌋

[SWS_CRYPT_40115] Definition of API function ara::crypto::x509::BasicCertInfo::GetConstraints

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Scope:	`class ara::crypto::x509::BasicCertInfo`	
Syntax:	`virtual KeyConstraints GetConstraints () const noexcept=0;`	
Return value:	KeyConstraints	key constraints
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the key constraints for the key associated with this PKCS#10 object.	

⌋

[SWS_CRYPT_40114] Definition of API function ara::crypto::x509::BasicCertInfo::GetPathLimit
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Scope:	`class ara::crypto::x509::BasicCertInfo`	
Syntax:	`virtual std::uint32_t GetPathLimit () const noexcept=0;`	
Return value:	std::uint32_t	certification path length limit
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the constraint on the path length defined in the Basic Constraints extension.	

⌋

[SWS_CRYPT_40113] Definition of API function ara::crypto::x509::BasicCertInfo::IsCa
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Scope:	`class ara::crypto::x509::BasicCertInfo`	
Syntax:	`virtual bool IsCa () const noexcept=0;`	
Return value:	bool	true if it is a CA request and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether the CA attribute of X509v3 Basic Constraints is true (i.e. pathlen=0).	

⌋

[SWS_CRYPT_40112] Definition of API function ara::crypto::x509::BasicCertInfo::SubjectDn
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Syntax:	`virtual const X509DN & SubjectDn () const noexcept=0;`

▽

△

Return value:	const X509DN &	subject DN
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the subject DN.	

⌋

[SWS_CRYPT_40111] Definition of API function ara::crypto::x509::BasicCertInfo::SubjectPubKey

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Scope:	`class ara::crypto::x509::BasicCertInfo`	
Syntax:	`virtual const X509PublicKeyInfo & SubjectPubKey (ara::core::Optional< cryp::CryptoProvider::Uptr > cryptoProvider) 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Load the subject public key information object to realm of specified crypto provider. If `(cryptoProvider is not provided)` then `X509PublicKeyInfo` object will be loaded in realm of the Stack-default Crypto Provider.	

⌋

[SWS_CRYPT_40217] Definition of API function ara::crypto::x509::Certificate::AuthorityKeyId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual ara::core::Result< std::size_t > AuthorityKeyId (ReadWriteMemRegion out) const noexcept=0;`	
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Get the DER encoded AuthorityKeyIdentifier of this certificate.	

⌋

[SWS_CRYPT_40215] Definition of API function ara::crypto::x509::Certificate::EndTime

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual time_t EndTime () const noexcept=0;`	
Return value:	time_t	"Not After" of the certificate
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the "Not After" of the certificate.	

⌋

[SWS_CRYPT_40220] Definition of API function ara::crypto::x509::Certificate::GetFingerprint

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual ara::core::Result< std::size_t > GetFingerprint (ReadWriteMemRegion 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kIncompleteArgState	--
		if the hashCtx context is not initialized

▽

△

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.

⌋

[SWS_CRYPT_40213] Definition of API function ara::crypto::x509::Certificate::IssuerDn

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual const X509DN & IssuerDn () const =0;`	
Return value:	const X509DN &	Issuer DN of this certificate
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Get the issuer certificate DN.	

⌋

[SWS_CRYPT_40216] Definition of API function ara::crypto::x509::Certificate::SerialNumber

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual ara::core::Result< std::size_t > SerialNumber (ReadWriteMemRegion out) const noexcept=0;`	
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Get the serial number of this certificate (should be 20 Bytes).	

⌋

[SWS_CRYPT_40214] Definition of API function ara::crypto::x509::Certificate::StartTime

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual time_t StartTime () const noexcept=0;`	
Return value:	time_t	"Not Before" of the certificate
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the "Not Before" of the certificate.	

⌋

[SWS_CRYPT_40218] Definition of API function ara::crypto::x509::Certificate::SubjectKeyId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual ara::core::Result< std::size_t > SubjectKeyId (ReadWriteMemRegion out) const noexcept=0;`	
Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
Description:	Get the DER encoded SubjectKeyIdentifier of this certificate.	

⌋

[SWS_CRYPT_40211] Definition of API function ara::crypto::x509::Certificate::X509Version

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/certificate.h"	
Scope:	`class ara::crypto::x509::Certificate`	
Syntax:	`virtual std::uint32_t X509Version () const noexcept=0;`	
Return value:	std::uint32_t	X.509 version
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the X.509 version of this certificate object.	

⌋

[SWS_CRYPT_40311] Definition of API function ara::crypto::x509::CertSignRequest::Verify

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/cert_sign_request.h"	
Scope:	`class ara::crypto::x509::CertSignRequest`	
Syntax:	`virtual bool Verify () const noexcept=0;`	
Return value:	bool	true if the signature is correct
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Verifies self-signed signature of the certificate request.	

⌋

[SWS_CRYPT_40313] Definition of API function ara::crypto::x509::CertSignRequest::ExportASN1CertSignRequest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Scope:	`class ara::crypto::x509::CertSignRequest`
Syntax:	`virtual ara::core::Result< std::size_t > ExportASN1CertSignRequest (ReadWriteMemRegion out) noexcept=0;`

▽

△

Parameters (out):	out	Output buffer
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidUsageOrder	--
		this error will be returned in case not all required information has been provided
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if out does not have sufficient capacity
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.	

⌋

[SWS_CRYPT_40314] Definition of API function ara::crypto::x509::CertSignRequest::Version

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/cert_sign_request.h"	
Scope:	`class ara::crypto::x509::CertSignRequest`	
Syntax:	`virtual unsigned Version () const noexcept=0;`	
Return value:	unsigned	format version of the certificate request
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return format version of the certificate request.	

⌋

[SWS_CRYPT_40711] Definition of API function ara::crypto::x509::OcspRequest::Version

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/ocsp_request.h"	
Scope:	`class ara::crypto::x509::OcspRequest`	
Syntax:	`virtual std::uint32_t Version () const noexcept=0;`	
Return value:	std::uint32_t	OCSP request format version
Exception Safety:	exception safe	

▽

△

Thread Safety:	thread-safe
Description:	Get version of the OCSP request format.

⌋

[SWS_CRYPT_40811] Definition of API function ara::crypto::x509::OcspResponse::Version

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/ocsp_response.h"	
Scope:	`class ara::crypto::x509::OcspResponse`	
Syntax:	`virtual std::uint32_t Version () const noexcept=0;`	
Return value:	std::uint32_t	OCSP response format version
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get version of the OCSP response format.	

⌋

[SWS_CRYPT_40413] Definition of API function ara::crypto::x509::X509DN::GetAttribute

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`virtual ara::core::Result< ara::core::String > GetAttribute (AttributeId id) const noexcept=0;`	
Parameters (in):	id	the identifier of required attribute
Return value:	ara::core::Result< ara::core::String >	String of the attribute
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the id argument has unsupported value
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output

▽

△

Description:	Get DN attribute by its ID (this method is applicale to all attributes except `kOrgUnit` 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.

⌋

[SWS_CRYPT_40415] Definition of API function ara::crypto::x509::X509DN::GetAttribute

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`virtual ara::core::Result< ara::core::String > GetAttribute (AttributeId id, unsigned 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the id argument has unsupported value
	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if (id != kOrgUnit) && (id != kDomainComponent) && index > 0)
	ara::crypto::CryptoErrc::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
Description:	Return DN attribute by its ID and sequential index (this method is applicale to attributes `kOrgUnit` 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.	

⌋

[SWS_CRYPT_40411] Definition of API function ara::crypto::x509::X509DN::GetDnString

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`virtual ara::core::Result< ara::core::String > GetDnString () const noexcept=0;`	
Return value:	ara::core::Result< ara::core::String >	String of the whole DN string
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInsufficientCapacity	--
		if (dn != nullptr), but dn->capacity() is less than required for the output value storing
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.	

⌋

[SWS_CRYPT_40417] Definition of API function ara::crypto::x509::X509DN::operator==

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`virtual bool operator== (const X509DN &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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check for equality of this and another Distinguished Name (DN) objects.	

⌋

[SWS_CRYPT_40418] Definition of API function ara::crypto::x509::X509DN::operator!=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`bool operator!= (const X509DN &other) const 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check for inequality of this and another Distinguished Name (DN) objects.	

⌋

[SWS_CRYPT_40414] Definition of API function ara::crypto::x509::X509DN::SetAttribute

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`virtual ara::core::Result< void > SetAttribute (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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the id argument has unsupported value
	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if the attribute string contains incorrect characters or it has unsupported length
Description:	Set DN attribute by its ID (this method is applicale to all `kDomainComponent`).	

⌋

[SWS_CRYPT_40416] Definition of API function ara::crypto::x509::X509DN::SetAttribute

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`virtual ara::core::Result< void > SetAttribute (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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the id argument has unsupported value
	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if the attribute string contains incorrect characters or it has unsupported length
	ara::crypto::CryptoErrc::kInvalidArgument	--
		if (id != kOrgUnit) && (id != kDomainComponent) && (index > 0)
	ara::crypto::CryptoErrc::kAboveBoundary	--
		if ((id == kOrgUnit) || (id == kDomainComponent)) and the index value is greater than the current number of components in the specified attribute
Description:	Set DN attribute by its ID and sequential index (this method is applicale to attributes `kOrgUnit` and `kDomainComponent`).	

⌋

[SWS_CRYPT_40412] Definition of API function ara::crypto::x509::X509DN::SetDn

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Syntax:	`virtual ara::core::Result< void > 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Set whole Distinguished Name (DN) from a single string. [Error]: ara::crypto::CryptoErrc::k UnexpectedValue if the `dn` string has incorrect syntax.

⌋

[SWS_CRYPT_40511] Definition of API function ara::crypto::x509::X509Extensions::Count

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_extensions.h"	
Scope:	`class ara::crypto::x509::X509Extensions`	
Syntax:	`virtual std::size_t Count () const noexcept=0;`	
Return value:	std::size_t	number of elements in the sequence
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Count number of elements in the sequence.	

⌋

[SWS_CRYPT_40911] Definition of API function ara::crypto::x509::X509Object::MyProvider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_object.h"	
Scope:	`class ara::crypto::x509::X509Object`	
Syntax:	`virtual X509Provider & MyProvider () const noexcept=0;`	
Return value:	X509Provider &	a reference to X.509 Provider instance that provides this object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get a reference to X.509 Provider of this object.	

⌋

[SWS_CRYPT_40612] Definition of API function ara::crypto::x509::X509Provider::BuildDn

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< X509DN::Uptrc > BuildDn (ara::core::String View dn) 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the dn argument has incorrect format
	ara::crypto::CryptoErrc::kInvalidInputSize	--
		if the dn argument has unsupported length (too large)
Description:	Create completed X.500 Distinguished Name structure from the provided string representation.	

⌋

[SWS_CRYPT_40629] Definition of API function ara::crypto::x509::X509Provider::CheckCertStatus

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual Certificate::Status CheckCertStatus (const Certificate &cert, const OcspResponse &ocspResponse, const Certificate &rootCert) const noexcept=0;`	
Parameters (in):	cert	a certificate that should be verified
	ocspResponse	an OCSP response
	rootCert	root certificate
Return value:	Certificate::Status	verification status of the provided certificate
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check certificate status by directly provided OCSP response. This method may be used for implementation of the "OCSP stapling".	

⌋

[SWS_CRYPT_40630] Definition of API function ara::crypto::x509::X509Provider::CheckCertStatus

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< bool > CheckCertStatus (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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the provided certificates are invalid
	ara::crypto::CryptoErrc::kRuntimeFault	--
		if the ocspResponse is invalid
Description:	Check status of a certificates list by directly provided OCSP response. This method may be used for implementation of the "OCSP stapling".	

⌋

[SWS_CRYPT_40635] Definition of API function ara::crypto::x509::X509Provider::CleanupVolatileStorage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`
Syntax:	`virtual void CleanupVolatileStorage () noexcept=0;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	thread-safe
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! .

⌋

[SWS_CRYPT_40640] Definition of API function ara::crypto::x509::X509Provider::CreateCertSignRequest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< CertSignRequest::Uptrc > CreateCertSignRequest (cryp::SignerPrivateCtx::Uptr signerCtx, ReadOnlyMemRegion derSubjectDN, ara::core::Optional< ReadOnlyMemRegion > x509Extensions, unsigned version) 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 might be included to the certificate signing request
	version	the format version of the target certificate signing request
Return value:	ara::core::Result< CertSignRequest::Uptrc >	unique smart pointer to created certificate signing request
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if any of arguments has incorrect/unsupported value
Description:	Create certificate signing request for a private key loaded to the context.	

⌋

[SWS_CRYPT_40615] Definition of API function ara::crypto::x509::X509Provider::CountCertsInChain

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< std::size_t > CountCertsInChain (ReadOnlyMemRegion certChain, ara::core::Optional< Serializable::FormatId > formatId) const noexcept=0;`	
Parameters (in):	certChain	DER/PEM-encoded certificate chain (in form of a single BLOB)
	formatId	input format identifier (Not providing it means auto-detect)
Return value:	ara::core::Result< std::size_t >	number of certificates in the chain
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the certChain argument cannot be pre-parsed
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the formatId argument has unknown value
Description:	Count number of certificates in a serialized certificate chain represented by a single BLOB.	

⌋

[SWS_CRYPT_40611] Definition of API function ara::crypto::x509::X509Provider::CreateEmptyDn

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< X509DN::Uptr > CreateEmptyDn (std::size_t capacity) 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Create an empty X.500 Distinguished Name (DN) structure. If `(0 == capacity)` then a maximally supported (by the implementation) capacity must be reserved.	

⌋

[SWS_CRYPT_40636] Definition of API function ara::crypto::x509::X509Provider::CreateEmptyExtensions

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< X509Extensions::Uptr > CreateEmptyExtensions (std::size_t capacity) 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 >	Unique smart pointer to created empty X509X509Extensions object
Exception Safety:	exception safe	

▽

△

Thread Safety:	thread-safe
Description:	Create an empty X.509 Extensions structure. If `(0 == capacity)` then a maximally supported (by the implementation) capacity must be reserved.

⌋

[SWS_CRYPT_40626] Definition of API function ara::crypto::x509::X509Provider::CreateOcspRequest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< OcspRequest::Uptrc > CreateOcspRequest (const Certificate &cert, ara::core::Optional< const cryp::Signer PrivateCtx::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< Ocsp Request::Uptrc >	unique smart pointer to the created OCSP request
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidArgument	--
		if the provided certificate is invalid
	ara::crypto::CryptoErrc::k IncompleteArgState	--
		if the signer context is not initialized by a key
Description:	Create OCSP request for specified certificate. This method may be used for implementation of the "OCSP stapling".	

⌋

[SWS_CRYPT_40627] Definition of API function ara::crypto::x509::X509Provider::CreateOcspRequest

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`

▽

△

Syntax:	`virtual ara::core::Result< OcspRequest::Uptrc > 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< Ocsp Request::Uptrc >	unique smart pointer to the created OCSP request
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidArgument	--
		if the provided certificates are invalid
	ara::crypto::CryptoErrc::k IncompleteArgState	--
		if the signer context is not initialized by a key
Description:	Create OCSP request for specified list of certificates. This method may be used for implementation of the "OCSP stapling".	

⌋

[SWS_CRYPT_40613] Definition of API function ara::crypto::x509::X509Provider::DecodeDn

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< X509DN::Uptrc > DecodeDn (ReadOnlyMemRegion dn, ara::core::Optional< Serializable::FormatId > formatId) noexcept=0;`	
Parameters (in):	dn	DER/PEM-encoded representation of the Distinguished Name
	formatId	input format identifier (Not providing it means auto-detect)
Return value:	ara::core::Result< X509DN::Uptrc >	unique smart pointer for the created X509DN object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::k InvalidArgument	--
		if the dn argument cannot be parsed
	ara::crypto::CryptoErrc::k UnknownIdentifier	--
		if the formatId argument has unknown value
Description:	Decode X.500 Distinguished Name structure from the provided serialized format.	

⌋

[SWS_CRYPT_40631] Definition of API function ara::crypto::x509::X509Provider::FindCertByDn

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Vector< Certificate::Uptrc > FindCertByDn (const X509DN &subjectDn, 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Find a certificate by the subject and issuer Distinguished Names (DN).	

⌋

[SWS_CRYPT_40632] Definition of API function ara::crypto::x509::X509Provider::FindCertByKeyIds

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Vector< Certificate::Uptrc > 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 kUnknownIdentifier
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Find a certificate by its SKID & AKID.	

⌋

[SWS_CRYPT_40633] Definition of API function ara::crypto::x509::X509Provider::FindCertBySn

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< Certificate::Uptrc > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the specified certificate could not be found
Description:	Find a certificate by its serial number and issue DN.	

⌋

[SWS_CRYPT_40634] Definition of API function ara::crypto::x509::X509Provider::ParseCertSignRequest

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< CertSignRequest::Uptrc > ParseCertSignRequest (ReadOnlyMemRegion csr, ara::core::Optional< bool > withMetaData) 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 CertSignRequest::ExportASN1CertSignRequest() interface
Return value:	ara::core::Result< CertSignRequest::Uptrc >	unique smart pointer to the certificate signing request
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupportedFormat	--
		is returned in case the provided buffer does not contain the expected format
Description:	Parse a certificate signing request (CSR) provided by the user.	

⌋

[SWS_CRYPT_40620] Definition of API function ara::crypto::x509::X509Provider::ImportCrl

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< void > ImportCrl (ReadOnlyMemRegion crl) noexcept=0;`	
Parameters (in):	crl	serialized CRL or Delta CRL (in form of a BLOB)
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if the provided BLOB is not a CRL/DeltaCRL
	ara::crypto::CryptoErrc::kRuntimeFault	--
		if the CRL validation has failed
Description:	Import Certificate Revocation List (CRL) or Delta CRL from a memory BLOB. The CRL has to be signed by a persistently stored certificate. Subsequent calls to VerifyCert() and VerifyCertChain() shall consider the imported CRL.	

⌋

[SWS_CRYPT_40621] Definition of API function ara::crypto::x509::X509Provider::Import

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< void > Import (const Certificate &cert, ara::core::Optional< ara::core::InstanceSpecifier > iSpecify) noexcept=0;`	
Parameters (in):	cert	a 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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kContentDuplication	--
		if the provided certificate already exists in the storage
	ara::crypto::CryptoErrc::kAccessViolation	--

▽

△

		if the InstanceSpecifier points to a CertificateSlot, which the application may only read
Violations:	ProcessMappingViolation	In case InstanceSpecifier does not point to a PortPrototype typed by a CryptoCertificateInterface modeled for the current process.
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.	

⌋

[SWS_CRYPT_40641] Definition of API function ara::crypto::x509::X509Provider::LoadCertificate

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< Certificate::Uptr > LoadCertificate (ara::core::InstanceSpecifier &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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnreservedResource	--
		if the InstanceSpecifier is incorrect (the certificate cannot be found)
Violations:	ProcessMappingViolation	In case InstanceSpecifier does not point to a PortPrototype typed by a CryptoCertificateInterface modeled for the current process.
Description:	Load a certificate from the persistent certificate storage.	

⌋

[SWS_CRYPT_40616] Definition of API function ara::crypto::x509::X509Provider::ParseCertChain

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`

▽

△

Syntax:	`virtual ara::core::Result< ara::core::Vector< Certificate::Uptr > > ParseCertChain (ReadOnlyMemRegion certChain, ara::core::Optional< Serializable::FormatId > formatId) noexcept=0;`	
Parameters (in):	certChain	DER/PEM-encoded certificate chain (in form of a single BLOB)
	formatId	input format identifier (Not providing it means auto-detect)
Return value:	ara::core::Result< ara::core::Vector< Certificate::Uptr > >	vector of certificates
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the certChain argument cannot be parsed
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the formatId argument has unknown value
Description:	Parse a serialized representation of the certificate chain and create their instances. Certificates in the returned vector will be placed in the same order as provided in `certChain`.	

⌋

[SWS_CRYPT_40617] Definition of API function ara::crypto::x509::X509Provider::ParseCertChain

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< ara::core::Vector< Certificate::Uptr > > ParseCertChain (const ara::core::Vector< ReadOnlyMemRegion > &certChain, ara::core::Optional< Serializable::FormatId > formatId) noexcept=0;`	
Parameters (in):	certChain	DER/PEM-encoded certificate chain in form of a vector containing individual certificates.
	formatId	input format identifier (Not providing it means auto-detect)
Return value:	ara::core::Result< ara::core::Vector< Certificate::Uptr > >	vector of certificates
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the certChain argument cannot be parsed
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the formatId argument has unknown value
Description:	Parse a serialized representation of the certificate chain and create their instances. Certificates in the returned vector will be placed in the same order as provided in `certChain`.	

⌋

[SWS_CRYPT_40614] Definition of API function ara::crypto::x509::X509Provider::ParseCert

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< Certificate::Uptr > ParseCert (ReadOnlyMemRegion cert, ara::core::Optional< Serializable::FormatId > formatId) noexcept=0;`	
Parameters (in):	cert	DER/PEM-encoded certificate
	formatId	input format identifier (Not providing it means auto-detect)
Return value:	ara::core::Result< Certificate::Uptr >	unique smart pointer to created certificate
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kInvalidArgument	--
		if the cert argument cannot be parsed
	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if the formatId argument has unknown value
Description:	Parse a serialized representation of the certificate and create a certificate object.	

⌋

[SWS_CRYPT_40628] Definition of API function ara::crypto::x509::X509Provider::ParseOcspResponse

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< OcspResponse::Uptrc > ParseOcspResponse (ReadOnlyMemRegion response) const noexcept=0;`	
Parameters (in):	response	a serialized OCSP response
Return value:	ara::core::Result< OcspResponse::Uptrc >	unique smart pointer to the created OCSP response instance
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnexpectedValue	--
		if the provided BLOB response doesn't keep an OCSP response
Description:	Parse serialized OCSP response and create correspondent interface instance. This method may be used for implementation of the "OCSP stapling".	

⌋

[SWS_CRYPT_40622] Definition of API function ara::crypto::x509::X509Provider::Remove

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual bool Remove (Certificate::Uptr cert) noexcept=0;`	
Parameters (in):	cert	a unique smart pointer to a certificate that should be removed
Return value:	bool	true if the certificate was found and removed from the storage, false if it was not found
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Remove specified certificate from the storage (volatile or persistent) and clear the certificate slot it was stored in.	

⌋

[SWS_CRYPT_40618] Definition of API function ara::crypto::x509::X509Provider::VerifyCert

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual Certificate::Status VerifyCert (const Certificate &cert, const Certificate &myRoot) noexcept=0;`	
Parameters (in):	cert	target certificate for verification
	myRoot	root certificate to be used for verification
Return value:	Certificate::Status	verification status of the provided certificate
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Verify the provided X509 certificate `cert` against the provided root certificate `myRoot`.	

⌋

[SWS_CRYPT_40619] Definition of API function ara::crypto::x509::X509Provider::VerifyCertChain

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual Certificate::Status VerifyCertChain (ara::core::Span< std::reference_wrapper< const Certificate > > chain) const noexcept=0;`	
Parameters (in):	chain	target certificate chain for verification
Return value:	Certificate::Status	verification status of the provided certificate chain
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Verify status of the provided certification chain. Verification status of the certificate chain is Certificate::Status::kValid only if the provided certificate chain can be validated according to rfc5280.	

⌋

[SWS_CRYPT_40914] Definition of API function ara::crypto::x509::X509Provider::ParseCustomCertExtensions

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< void > ParseCustomCertExtensions (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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		If parsing the extensions fails or calling one of the callback returns an error.
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.	

⌋

[SWS_CRYPT_40915] Definition of API function ara::crypto::x509::X509Provider::ParseCustomCertExtensions

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Scope:	`class ara::crypto::x509::X509Provider`	
Syntax:	`virtual ara::core::Result< void > ParseCustomCertExtensions (const Certificate &cert, std::unique_ptr< X509CustomExtensionsParser > customExtensionsParser, X509CustomExtensionsParser::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 >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		If parsing the extensions fails or calling one of the callback returns an error.
	ara::crypto::CryptoErrc::kUnexpectedValue	--
		If the certificate doesn't contain an extension with the provided Oid.
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 custom ExtensionsParser 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.	

⌋

[SWS_CRYPT_40604] Definition of API function ara::crypto::x509::X509Provider::~X509Provider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`
Syntax:	`virtual ~X509Provider () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_30226] Definition of API function ara::crypto::x509::X509Provider::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`
Syntax:	`X509Provider & operator= (const X509Provider &other)=delete;`
Description:	Copy-assign another X509Provider to this instance.

⌋

[SWS_CRYPT_30227] Definition of API function ara::crypto::x509::X509Provider::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`
Syntax:	`X509Provider & operator= (X509Provider &&other)=delete;`
Description:	Move-assign another X509Provider to this instance.

⌋

[SWS_CRYPT_41013] Definition of API function ara::crypto::x509::X509Provider::X509Provider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`
Syntax:	`X509Provider (const X509Provider &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41014] Definition of API function ara::crypto::x509::X509Provider::X509Provider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`
Syntax:	`X509Provider (X509Provider &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_40922] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnBitString

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnBitString (BitString parsed_bit_string) noexcept=0;`	
Parameters (in):	parsed_bit_string	Parsed bit string value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a bit string is encountered.	

⌋

[SWS_CRYPT_40920] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnBool

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`

▽

△

Syntax:	`virtual ara::core::Result< void > OnBool (bool parsed_bool) noexcept=0;`	
Parameters (in):	parsed_bool	Parsed boolean value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a boolean is encountered.	

⌋

[SWS_CRYPT_40929] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnGeneralizedTime

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnGeneralizedTime (GeneralizedTime parsed_generalized_time) noexcept=0;`	
Parameters (in):	parsed_generalized_time	Parsed generalized time value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a generalized time is encountered.	

⌋

[SWS_CRYPT_40928] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnIa5String

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Syntax:	`virtual ara::core::Result< void > OnIa5String (Ia5String parsed_ia5_string) noexcept=0;`

▽

△

Parameters (in):	parsed_ia5_string	Parsed IA5 string value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when an IA5 string is encountered.	

⌋

[SWS_CRYPT_40921] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnInteger

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnInteger (Integer parsed_integer) noexcept=0;`	
Parameters (in):	parsed_integer	Parsed integer value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when an integer is encountered.	

⌋

[SWS_CRYPT_40924] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnNull

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnNull () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	

▽

△

Errors:	ara::crypto::CryptoErrc::kRuntimeFault	-- Indicates an error to the parser to stop parsing
Description:	Called when a NULL is encountered.	

⌋

[SWS_CRYPT_40923] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnOctetString

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnOctetString (OctetString parsed_octet_string) noexcept=0;`	
Parameters (in):	parsed_octet_string	Parsed octet string value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	-- Indicates an error to the parser to stop parsing
Description:	Called when an octet string is encountered.	

⌋

[SWS_CRYPT_40925] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnOid

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnOid (Oid parsed_oid) noexcept=0;`	
Parameters (in):	parsed_oid	Parsed oid value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	-- Indicates an error to the parser to stop parsing
Description:	Called when an oid is encountered.	

⌋

[SWS_CRYPT_40931] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnParsingEnd

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnParsingEnd () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when the parsing is completed.	

⌋

[SWS_CRYPT_40927] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnPrintableString

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnPrintableString (PrintableString parsed_printable_string) noexcept=0;`	
Parameters (in):	parsed_printable_string	Parsed printable string value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a printable string is encountered.	

⌋

[SWS_CRYPT_40917] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnSequenceEnd

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnSequenceEnd () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a sequence ends.	

⌋

[SWS_CRYPT_40916] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnSequenceStart

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnSequenceStart () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a sequence starts.	

⌋

[SWS_CRYPT_40919] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnSetEnd

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnSetEnd () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a set ends.	

⌋

[SWS_CRYPT_40918] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnSetStart

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnSetStart () noexcept=0;`	
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a set starts.	

⌋

[SWS_CRYPT_40930] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnUtcTime
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnUtcTime (UtcTime parsed_utc_time) noexcept=0;`	
Parameters (in):	parsed_utc_time	Parsed UTC time value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when a UTC time is encountered.	

⌋

[SWS_CRYPT_40926] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::OnUtf8String
Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function	
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"	
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`	
Syntax:	`virtual ara::core::Result< void > OnUtf8String (Utf8String parsed_utf8_string) noexcept=0;`	
Parameters (in):	parsed_utf8_string	Parsed UTF8 string value
Return value:	ara::core::Result< void >	either a void return or an error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ara::crypto::CryptoErrc::kRuntimeFault	--
		Indicates an error to the parser to stop parsing
Description:	Called when an UTF8 string is encountered.	

⌋

[SWS_CRYPT_40981] Definition of API function ara::crypto::x509::X509CustomExtensionsParser::~X509CustomExtensionsParser

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	function
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Syntax:	`virtual ~X509CustomExtensionsParser () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_40101] Definition of API type ara::crypto::x509::BasicCertInfo::KeyConstraints

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	KeyConstraints
Syntax:	`using KeyConstraints = std::uint32_t;`
Description:	X.509 v3 Key Constraints type definition.

⌋

[SWS_CRYPT_40203] Definition of API enum ara::crypto::x509::Certificate::Status

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	enumeration
Header file:	#include "ara/crypto/x509/certificate.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`class ara::crypto::x509::Certificate`
Symbol:	Status
Underlying type:	std::uint32_t
Syntax:	`enum class Status : std::uint32_t {...};`

▽

△

Values:	kValid= 0	The signature of the provided certificate is successfully verified and the signing certificate is a root of trust or is chained to a root of trust on this adaptive machine (e.g. ECU).
	kInvalid= 1	The certificate is invalid e.g. the provided certificate can be invalid if the signature of the provided certificate cannot be verified by the root certificate.
	kNotAvailable= 3	A verification result is not available because verification could not be executed e.g. because the provided root is not the signing certificate or a root certificate could not be found.
	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.
	kRevoked= 6	The certificate has been revoked i.e. the provided certificate is on CRL list
Description:	Certificate verification status.	

⌋

[SWS_CRYPT_40202] Definition of API type ara::crypto::x509::Certificate::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/certificate.h"
Scope:	`class ara::crypto::x509::Certificate`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const Certificate>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40201] Definition of API type ara::crypto::x509::Certificate::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/certificate.h"
Scope:	`class ara::crypto::x509::Certificate`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<Certificate>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40301] Definition of API type ara::crypto::x509::CertSignRequest::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Scope:	`class ara::crypto::x509::CertSignRequest`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const CertSignRequest>;`
Description:	Unique smart pointer of the constant interface.

⌋

[SWS_CRYPT_40302] Definition of API type ara::crypto::x509::CertSignRequest::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Scope:	`class ara::crypto::x509::CertSignRequest`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<CertSignRequest>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40002] Definition of API enum ara::crypto::x509::OcspCertStatus

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/x509/ocsp_response.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`namespace ara::crypto::x509`	
Symbol:	OcspCertStatus	
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.
Description:	On-line Certificate Status Protocol (OCSP) Certificate Status.	

⌋

[SWS_CRYPT_40702] Definition of API type ara::crypto::x509::OcspRequest::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Scope:	`class ara::crypto::x509::OcspRequest`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const OcspRequest>;`
Description:	Unique smart pointer of a constant interface instance.

⌋

[SWS_CRYPT_40701] Definition of API type ara::crypto::x509::OcspRequest::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Scope:	`class ara::crypto::x509::OcspRequest`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<OcspRequest>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40001] Definition of API enum ara::crypto::x509::OcspResponseStatus

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/x509/ocsp_response.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`namespace ara::crypto::x509`	
Symbol:	OcspResponseStatus	
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.
Description:	On-line Certificate Status Protocol (OCSP) Response Status.	

⌋

[SWS_CRYPT_40802] Definition of API type ara::crypto::x509::OcspResponse::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Scope:	`class ara::crypto::x509::OcspResponse`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const OcspResponse>;`
Description:	Unique smart pointer of a constant interface instance.

⌋

[SWS_CRYPT_40801] Definition of API type ara::crypto::x509::OcspResponse::Uptr

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Scope:	`class ara::crypto::x509::OcspResponse`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<OcspResponse>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40403] Definition of API enum ara::crypto::x509::X509DN::AttributeId

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`class ara::crypto::x509::X509DN`	
Symbol:	AttributeId	
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.
Description:	Enumeration of DN attributes' identifiers.	

⌋

[SWS_CRYPT_40402] Definition of API type ara::crypto::x509::X509DN::Uptrc

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_dn.h"
Scope:	`class ara::crypto::x509::X509DN`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const X509DN>;`
Description:	Unique smart pointer of the constant interface.

⌋

[SWS_CRYPT_40401] Definition of API type ara::crypto::x509::X509DN::Uptr

Status: DRAFT

Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_dn.h"
Scope:	`class ara::crypto::x509::X509DN`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<X509DN>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40501] Definition of API type ara::crypto::x509::X509Extensions::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_extensions.h"
Scope:	`class ara::crypto::x509::X509Extensions`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<X509Extensions>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_24401] Definition of API type ara::crypto::x509::X509PublicKeyInfo::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02307

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"
Scope:	`class ara::crypto::x509::X509PublicKeyInfo`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const X509PublicKeyInfo>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40601] Definition of API type ara::crypto::x509::X509Provider::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_provider.h"
Scope:	`class ara::crypto::x509::X509Provider`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<X509Provider>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_40935] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::BitString

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	BitString
Syntax:	`using BitString = std::pair<ara::crypto::ReadOnlyMemRegion, NumberOfUnusedBits>;`
Description:	Type alias.

⌋

[SWS_CRYPT_40941] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::GeneralizedTime

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	GeneralizedTime
Syntax:	`using GeneralizedTime = ara::core::StringView;`
Description:	Type alias.

⌋

[SWS_CRYPT_40940] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::Ia5String

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	Ia5String
Syntax:	`using Ia5String = ara::core::StringView;`
Description:	Type alias.

⌋

[SWS_CRYPT_40933] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::Integer

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	Integer
Syntax:	`using Integer = ara::crypto::ReadOnlyMemRegion;`
Description:	Type alias.

⌋

[SWS_CRYPT_40934] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::NumberOfUnusedBits

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	NumberOfUnusedBits
Syntax:	`using NumberOfUnusedBits = std::uint8_t;`
Description:	Type alias.

⌋

[SWS_CRYPT_40936] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::OctetString

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	OctetString
Syntax:	`using OctetString = ara::crypto::ReadOnlyMemRegion;`
Description:	Type alias.

⌋

[SWS_CRYPT_40937] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::Oid

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	Oid
Syntax:	`using Oid = ara::core::StringView;`
Description:	Type alias.

⌋

[SWS_CRYPT_40939] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::PrintableString

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	PrintableString
Syntax:	`using PrintableString = ara::core::StringView;`
Description:	Type alias.

⌋

[SWS_CRYPT_40942] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::UtcTime

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	UtcTime
Syntax:	`using UtcTime = ara::core::StringView;`
Description:	Type alias.

⌋

[SWS_CRYPT_40938] Definition of API type ara::crypto::x509::X509CustomExtensionsParser::Utf8String

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	type alias
Header file:	#include "ara/crypto/x509/x509_custom_extension_parser.h"
Scope:	`class ara::crypto::x509::X509CustomExtensionsParser`
Symbol:	Utf8String
Syntax:	`using Utf8String = ara::crypto::ReadOnlyMemRegion;`
Description:	Type alias.

⌋

[SWS_CRYPT_40157] Definition of API variable ara::crypto::x509::BasicCert Info::kConstrCrlSign

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrCrlSign
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrCrlSign {0x0200};`
Description:	The key can be used for Certificates Revokation Lists (CRL) signing.

⌋

[SWS_CRYPT_40154] Definition of API variable ara::crypto::x509::BasicCert Info::kConstrDataEncipherment

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrDataEncipherment
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrDataEncipherment {0x1000};`
Description:	The key can be used for data encipherment.

⌋

[SWS_CRYPT_40159] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrDecipherOnly

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrDecipherOnly
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrDecipherOnly {0x0080};`
Description:	The enciphermet key can be used for deciphering only.

⌋

[SWS_CRYPT_40151] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrDigitalSignature

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrDigitalSignature
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrDigitalSignature {0x8000};`
Description:	The key can be used for digital signature production.

⌋

[SWS_CRYPT_40158] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrEncipherOnly

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrEncipherOnly
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrEncipherOnly {0x0100};`

▽

△

Description:	The enciphermet key can be used for enciphering only.

⌋

[SWS_CRYPT_40155] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrKeyAgreement

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrKeyAgreement
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrKeyAgreement {0x0800};`
Description:	The key can be used for a key agreement protocol execution.

⌋

[SWS_CRYPT_40156] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrKeyCertSign

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrKeyCertSign
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrKeyCertSign {0x0400};`
Description:	The key can be used for certificates signing.

⌋

[SWS_CRYPT_40153] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrKeyEncipherment

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrKeyEncipherment
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrKeyEncipherment {0x2000};`
Description:	The key can be used for key encipherment.

⌋

[SWS_CRYPT_40152] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrNonRepudiation

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrNonRepudiation
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrNonRepudiation {0x4000};`
Description:	The key can be used in cases requiring the "non-repudiation" guarantee.

⌋

[SWS_CRYPT_40150] Definition of API variable ara::crypto::x509::BasicCertInfo::kConstrNone

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306

⌈

Kind:	variable
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Scope:	`class ara::crypto::x509::BasicCertInfo`
Symbol:	kConstrNone
Type:	`const KeyConstraints`
Syntax:	`static const KeyConstraints kConstrNone {0};`

▽

△

Description:	No key constraints.

⌋

8.4 API Common Data Types

[SWS_CRYPT_10015] Definition of API type ara::crypto::AllowedUsageFlags

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	AllowedUsageFlags
Syntax:	`using AllowedUsageFlags = std::uint32_t;`
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 `kAllowKeyDerivation`!

⌋

[SWS_CRYPT_10014] Definition of API type ara::crypto::CryptoAlgId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02102, RS_CRYPTO_02107

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	CryptoAlgId
Syntax:	`using CryptoAlgId = std::uint64_t;`
Description:	Container type of the Crypto Algorithm Identifier.

⌋

[SWS_CRYPT_10016] Definition of API enum ara::crypto::CryptoObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/common/base_id_types.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`namespace ara::crypto`	
Symbol:	CryptoObjectType	
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
	kPrivateKey= 2	`cryp::PrivateKey` object
	kPublicKey= 3	`cryp::PublicKey` object
	kSecretSeed= 4	`cryp::SecretSeed` object. **Note:** the seed cannot have an associated crypto algorithm!
Description:	Enumeration of all types of crypto objects, i.e. types of content that can be stored to a key slot.	

⌋

[SWS_CRYPT_10100] Definition of API class ara::crypto::CryptoObjectUid

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005, RS_CRYPTO_02006

⌈

Kind:	struct
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto`
Symbol:	CryptoObjectUid
Syntax:	`struct CryptoObjectUid {...};`
Description:	Definition of Crypto Object Unique Identifier (**COUID**) type.

⌋

[SWS_CRYPT_10017] Definition of API enum ara::crypto::ProviderType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401, RS_CRYPTO_02109

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/common/base_id_types.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`namespace ara::crypto`	
Symbol:	ProviderType	
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.
Description:	Enumeration of all known Provider types.	

⌋

[SWS_CRYPT_10033] Definition of API type ara::crypto::ReadOnlyMemRegion

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/mem_region.h"
Scope:	`namespace ara::crypto`
Symbol:	ReadOnlyMemRegion
Syntax:	`using ReadOnlyMemRegion = ara::core::Span<const std::uint8_t>;`
Description:	Read-Only Memory Region (intended for [in] arguments)

⌋

[SWS_CRYPT_10031] Definition of API type ara::crypto::ReadWriteMemRegion

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/mem_region.h"
Scope:	`namespace ara::crypto`
Symbol:	ReadWriteMemRegion

▽

△

Syntax:	`using ReadWriteMemRegion = ara::core::Span<std::uint8_t>;`
Description:	Read-Write Memory Region (intended for [in/out] arguments)

⌋

[SWS_CRYPT_10099] Definition of API enum ara::crypto::CryptoErrc

Status: DRAFT

Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/common/crypto_error_domain.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`namespace ara::crypto`	
Symbol:	CryptoErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class CryptoErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kResourceFault= 1 * 0x1000000U	ResourceException: Generic resource fault!
	kBusyResource= k ResourceFault + 1	ResourceException: Specified resource is busy!
	kUnreservedResource= kResourceFault + 3	ResourceException: Specified resource was not reserved!
	kModifiedResource= k ResourceFault + 4	ResourceException: Specified resource has been modified!
	kInvalidArgument= (2U * 0x1000000U) + 1 * 0x10000U	InvalidArgumentException: An invalid argument value is provided!
	kUnknownIdentifier= k InvalidArgument + 1	InvalidArgumentException: Unknown identifier is provided!
	kInsufficientCapacity= k InvalidArgument + 2	InvalidArgumentException: Insufficient capacity of the output buffer!
	kInvalidInputSize= k InvalidArgument + 3	InvalidArgumentException: Invalid size of an input buffer!
	kIncompatible Arguments= kInvalid Argument + 4	InvalidArgumentException: Provided values of arguments are incompatible!
	kBelowBoundary= k InvalidArgument + 6	InvalidArgumentException: Provided value is below the lower boundary!
	kAboveBoundary= k InvalidArgument + 7	InvalidArgumentException: Provided value is above the upper boundary!
	kAuthTagNotValid= k InvalidArgument + 8	AuthTagNotValidException: Provided authentication-tag cannot be verified!
	kUnsupported= kInvalid Argument + 1 * 0x100U	UnsupportedException: Unsupported request (due to limitations of the implementation)!
	kInvalidUsageOrder= (2U * 0x1000000U) + 2 * 0x10000U	InvalidUsageOrderException: Invalid usage order of the interface!
	kUninitializedContext= k InvalidUsageOrder + 1	InvalidUsageOrderException: Context of the interface was not initialized!

▽

△

	kProcessingNotStarted= kInvalidUsageOrder + 2	InvalidUsageOrderException: Data processing was not started yet!
	kProcessingNot Finished= kInvalidUsage Order + 3	InvalidUsageOrderException: Data processing was not finished yet!
	kRuntimeFault= 3 * 0x1000000U	RuntimeException: Generic runtime fault!
	kUnsupportedFormat= k RuntimeFault + 1	RuntimeException: Unsupported serialization format for this object type!
	kBruteForceRisk= k RuntimeFault + 2	RuntimeException: Operation is prohibitted due to a risk of a brute force attack!
	kContentRestrictions= k RuntimeFault + 3	RuntimeException: The operation violates content restrictions of the target container!
	kContentDuplication= k RuntimeFault + 6	RuntimeException: Provided content already exists in the target storage!
	kUnexpectedValue= k RuntimeFault + 1 * 0x10000U	UnexpectedValueException: Unexpected value of an argument is provided!
	kIncompatibleObject= k UnexpectedValue + 1	UnexpectedValueException: The provided object is incompatible with requested operation or its configuration!
	kIncompleteArgState= k UnexpectedValue + 2	UnexpectedValueException: Incomplete state of an argument!
	kEmptyContainer= k UnexpectedValue + 3	UnexpectedValueException: Specified container is empty!
	kMissingArgument= k UnexpectedValue + 4	kMissingArgumentException: Expected argument, but none provided!
	kBadObjectType= k UnexpectedValue + 1 * 0x100U	BadObjectTypeException: Provided object has unexpected type!
	kUsageViolation= k RuntimeFault + 2 * 0x10000U	UsageViolationException: Violation of allowed usage for the object!
	kAccessViolation= k RuntimeFault + 3 * 0x10000U	AccessViolationException: Access rights violation!
Description:	Enumeration of all Crypto Error Code values that may be reported by `ara::crypto`.	

⌋

[SWS_CRYPT_30001] Definition of API class ara::crypto::SecureCounter

Status: DRAFT

Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	struct
Header file:	#include "ara/crypto/common/entry_point.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto`
Symbol:	SecureCounter
Syntax:	`struct SecureCounter {...};`

▽

△

Description:	128 bit secure counter made up of most significant and least significant quad-word of the hardware counter.

⌋

[SWS_CRYPT_10701] Definition of API enum ara::crypto::Serializable::FormatId

Status: DRAFT

Upstream requirements: RS_CRYPTO_02004, RS_CRYPTO_02302

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/common/serializable.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`class ara::crypto::Serializable`	
Symbol:	FormatId	
Underlying type:	std::uint32_t	
Syntax:	`enum class FormatId : std::uint32_t {...};`	
Values:	kFormatP1363= 0	ISO/IEC 7816-8 / IEEE P1363 - this is a raw data encoding, i.e. r | s for ECDSA.
	kFormatPemEncoded= 1	rfc7468 PEM format (see also PKCS#7, CMS/rfc5652), i.e. SignatureValue ::= OCTET STRING, a length encoded byte-array, sometimes referred to as "DER-encoded"
	kFormat X509ASN1Encoded= 2	x.509 PKI / rfc3279 i.e. SEQUENCE := { // 1+2 INTEGER r; // 1+2+COUNTOF(r)+1 INTEGER s; } // 1+2+COUNTOF(s)+1
Description:	A container type for the encoding format identifiers.	

⌋

[SWS_CRYPT_10019] Definition of API enum ara::crypto::CryptoTransform

Status: DRAFT

Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/common/base_id_types.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`namespace ara::crypto`	
Symbol:	CryptoTransform	
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
Description:	Enumeration of cryptographic transformations.	

⌋

[SWS_CRYPT_10852] Definition of API type ara::crypto::VolatileTrustedContainer::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Scope:	`class ara::crypto::VolatileTrustedContainer`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<VolatileTrustedContainer>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_10400] Definition of API class ara::crypto::Uuid

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	struct
Header file:	#include "ara/crypto/common/uuid.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto`
Symbol:	Uuid
Syntax:	`struct Uuid {...};`
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!

⌋

[SWS_CRYPT_10801] Definition of API type ara::crypto::IOInterface::Uptr

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`
Symbol:	Uptr
Syntax:	`using Uptr = std::unique_ptr<IOInterface>;`
Description:	Unique smart pointer of the interface.

⌋

[SWS_CRYPT_10802] Definition of API type ara::crypto::IOInterface::Uptrc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`
Symbol:	Uptrc
Syntax:	`using Uptrc = std::unique_ptr<const IOInterface>;`
Description:	Unique smart pointer of the constant interface.

⌋

[SWS_CRYPT_19903] Definition of API type ara::crypto::CryptoErrorDomain::Errc

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/crypto_error_domain.h"
Scope:	`class ara::crypto::CryptoErrorDomain`
Symbol:	Errc
Syntax:	`using Errc = CryptoErrc;`
Description:	crypto error

⌋

[SWS_CRYPT_19904] Definition of API type ara::crypto::CryptoErrorDomain::Exception

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	type alias
Header file:	#include "ara/crypto/common/crypto_error_domain.h"
Scope:	`class ara::crypto::CryptoErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = CryptoException;`
Description:	Alias for the exception base class.

⌋

[SWS_CRYPT_10018] Definition of API enum ara::crypto::KeySlotType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	enumeration	
Header file:	#include "ara/crypto/common/base_id_types.h"	
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"	
Scope:	`namespace ara::crypto`	
Symbol:	KeySlotType	
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
Description:	Enumeration of key-slot types; currently only machine and applicaiton key-slots are defined.	

⌋

8.5 API Reference

[SWS_CRYPT_10800] Definition of API class ara::crypto::IOInterface

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	class
Header file:	#include "ara/crypto/common/io_interface.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto`
Symbol:	IOInterface
Syntax:	`class IOInterface {...};`
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.

⌋

[SWS_CRYPT_10700] Definition of API class ara::crypto::Serializable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02105

⌈

Kind:	class
Header file:	#include "ara/crypto/common/serializable.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto`
Symbol:	Serializable
Syntax:	`class Serializable {...};`
Description:	Serializable object interface.

⌋

[SWS_CRYPT_10850] Definition of API class ara::crypto::VolatileTrustedContainer

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	class
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"

▽

△

Scope:	`namespace ara::crypto`
Symbol:	VolatileTrustedContainer
Syntax:	`class VolatileTrustedContainer {...};`
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.

⌋

[SWS_CRYPT_19905] Definition of API class ara::crypto::CryptoException

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	class
Header file:	#include "ara/crypto/common/crypto_error_domain.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto`
Symbol:	CryptoException
Base class:	ara::core::Exception
Syntax:	`class CryptoException : public ara::core::Exception {...};`
Description:	Exception type thrown for CRYPTO errors.

⌋

[SWS_CRYPT_19900] Definition of API class ara::crypto::CryptoErrorDomain

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/crypto/common/crypto_error_domain.h"
Forwarding header file:	#include "ara/crypto/crypto_fwd.h"
Scope:	`namespace ara::crypto`
Symbol:	CryptoErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	`class CryptoErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	As per `ara::crypto::CryptoErrorDomain` in [SWS_CORE_90023]
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. .

⌋

[SWS_CRYPT_19951] Definition of API function ara::crypto::MakeErrorCode
Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_error_domain.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (CryptoErrorDomain::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:	exception safe	
Thread Safety:	implementation defined	
Description:	Makes Error Code instances from the Crypto Error Domain. The returned `ErrorCode` instance always references to `CryptoErrorDomain`.	

⌋

[SWS_CRYPT_19952] Definition of API function ara::crypto::GetCryptoErrorDomain
Status: DRAFT
Upstream requirements: SWS_CORE_10980

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_error_domain.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr const ara::core::ErrorDomain & GetCryptoErrorDomain () noexcept;`	
Return value:	const ara::core::ErrorDomain &	the CryptoErrorDomain
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Return a reference to the global CryptoErrorDomain.	

⌋

[SWS_CRYPT_20099] Definition of API function ara::crypto::LoadCryptoProvider

Status: DRAFT

Upstream requirements: RS_CRYPTO_02401, RS_CRYPTO_02301

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/entry_point.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`cryp::CryptoProvider::Uptr LoadCryptoProvider (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:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ProcessMappingViolation`	In case InstanceSpecifier does not point to a PortPrototype typed by a CryptoProviderInterface modeled for the current process.
Description:	Factory that creates or return existing single instance of specific Crypto Provider. If `(providerUid == nullptr)` then platform default provider should be loaded.	

⌋

[SWS_CRYPT_30099] Definition of API function ara::crypto::LoadKeyStorageProvider

Status: DRAFT

Upstream requirements: RS_CRYPTO_02109, RS_CRYPTO_02401, RS_CRYPTO_02301

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/entry_point.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`keys::KeyStorageProvider::Uptr LoadKeyStorageProvider () noexcept;`	
Return value:	ara::crypto::keys::KeyStorageProvider::Uptr	unique smart pointer to loaded Key Storage Provider
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Factory that creates or return existing single instance of the Key Storage Provider.	

⌋

[SWS_CRYPT_40099] Definition of API function ara::crypto::LoadX509Provider

Status: DRAFT
Upstream requirements: RS_CRYPTO_02306, RS_CRYPTO_02301

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/entry_point.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`x509::X509Provider::Uptr LoadX509Provider () noexcept;`	
Return value:	ara::crypto::x509::X509Provider::Uptr	unique smart pointer to loaded X.509 Provider
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_20098] Definition of API function ara::crypto::GetSecureCounter

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/entry_point.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`ara::core::Result< SecureCounter > 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:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupported	--
		if the Secure Counter is unsupported by the Crypto Stack implementation on this Platform
	ara::crypto::CryptoErrc::kAccessViolation	--
		if current Actor has no permission to call this routine
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!	

⌋

[SWS_CRYPT_10112] Definition of API function ara::crypto::CryptoObjectUid::HasEarlierVersionThan
Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`struct ara::crypto::CryptoObjectUid`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether this identifier was generated earlier than the one provided by the argument.	

⌋

[SWS_CRYPT_10113] Definition of API function ara::crypto::CryptoObjectUid::HasLaterVersionThan
Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`struct ara::crypto::CryptoObjectUid`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether this identifier was generated later than the one provided by the argument.	

⌋

[SWS_CRYPT_10111] Definition of API function ara::crypto::CryptoObjectUid::HasSameSourceAs

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`struct ara::crypto::CryptoObjectUid`	
Syntax:	`constexpr bool 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 m GeneratorUid field)
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether this identifier has a common source with the one provided by the argument.	

⌋

[SWS_CRYPT_10114] Definition of API function ara::crypto::CryptoObjectUid::IsNil

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`struct ara::crypto::CryptoObjectUid`	
Syntax:	`bool IsNil () const noexcept;`	
Return value:	bool	true if mGeneratorUid is "Nil" and mVersionStamp is 0, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether this CryptoObjectUid is not valid ("Nil").	

⌋

[SWS_CRYPT_10115] Definition of API function ara::crypto::CryptoObjectUid::SourceIsNil
Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`struct ara::crypto::CryptoObjectUid`	
Syntax:	`bool SourceIsNil () const noexcept;`	
Return value:	bool	true if this identifier is "Nil" and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether this object's generator identifier is "Nil".	

⌋

[SWS_CRYPT_10810] Definition of API function ara::crypto::IOInterface::~IOInterface
Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`
Syntax:	`virtual ~IOInterface () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_10819] Definition of API function ara::crypto::IOInterface::GetAllowedUsage
Status: DRAFT
Upstream requirements: RS_CRYPTO_02008

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual AllowedUsageFlags GetAllowedUsage () const noexcept=0;`	
Return value:	AllowedUsageFlags	allowed key/seed usage flags

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
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.

⌋

[SWS_CRYPT_10813] Definition of API function ara::crypto::IOInterface::GetCapacity

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual std::size_t GetCapacity () const noexcept=0;`	
Return value:	std::size_t	capacity of the underlying buffer of this IOInterface (in bytes)
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return capacity of the underlying resource.	

⌋

[SWS_CRYPT_10812] Definition of API function ara::crypto::IOInterface::GetCryptoObjectType

Status: DRAFT
Upstream requirements: RS_CRYPTO_02110

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual CryptoObjectType GetCryptoObjectType () const noexcept=0;`	
Return value:	CryptoObjectType	the CryptoObjectType stored inside the referenced resource
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the CryptoObjectType of the object referenced by this IOInterface.	

⌋

[SWS_CRYPT_10811] Definition of API function ara::crypto::IOInterface::GetObjectId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual CryptoObjectUid GetObjectId () const noexcept=0;`	
Return value:	CryptoObjectUid	COUID of an object stored in the container
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return COUID of an object stored to this IOInterface. Unambiguous identification of a crypto object requires both components: `CryptoObjectUid` and `CryptoObjectType`.	

⌋

[SWS_CRYPT_10817] Definition of API function ara::crypto::IOInterface::GetPayloadSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual std::size_t 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:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_10822] Definition of API function ara::crypto::IOInterface::GetPrimitiveId

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual ara::core::Result< CryptoAlgId > GetPrimitiveId () const noexcept=0;`	
Return value:	ara::core::Result< CryptoAlgId >	the binary Crypto Primitive ID
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kEmptyContainer	--
		If the underlying resource this IOInterface points to is empty
Description:	Get vendor specific ID of the primitive.	

⌋

[SWS_CRYPT_10818] Definition of API function ara::crypto::IOInterface::GetTypeRestriction

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004, RS_CRYPTO_02110

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual CryptoObjectType GetTypeRestriction () const noexcept=0;`	
Return value:	CryptoObjectType	an object type of allowed content (CryptoObjectType::kUndefined means without restriction)
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return content type restriction of this IOInterface. If KeySlotPrototypeProps::mAllowContentTypeChange==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.	

⌋

[SWS_CRYPT_10816] Definition of API function ara::crypto::IOInterface::IsObjectExportable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual bool IsObjectExportable () const noexcept=0;`	
Return value:	bool	true if an object stored to the container has set the "exportable" attribute
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_10815] Definition of API function ara::crypto::IOInterface::IsObjectSession

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual bool IsObjectSession () const noexcept=0;`	
Return value:	bool	true if the object referenced by this IOInterface has set the "session" attribute
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the "session" (or "temporary") attribute of an object as set e.g. by KeyDerivationFunctionCtx::DeriveKey(). A "session" object can be stored to a VolatileTrustedContainer only! If this IOInterface is linked to a KeySlot this returns always false.	

⌋

[SWS_CRYPT_10814] Definition of API function ara::crypto::IOInterface::IsVolatile

Status: DRAFT
Upstream requirements: RS_CRYPTO_02109

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual bool 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:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_10823] Definition of API function ara::crypto::IOInterface::IsValid

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/io_interface.h"	
Scope:	`class ara::crypto::IOInterface`	
Syntax:	`virtual bool IsValid () const noexcept=0;`	
Return value:	bool	true if the underlying resource can be valid, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
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.	

⌋

[SWS_CRYPT_10821] Definition of API function ara::crypto::IOInterface::IsWritable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`

▽

△

Syntax:	`virtual bool IsWritable () const noexcept=0;`	
Return value:	bool	true if the underlying resource can be written
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get whether the underlying KeySlot is writable - if this IOInterface is linked to a VolatileTrusted Container always return true.	

⌋

[SWS_CRYPT_30202] Definition of API function ara::crypto::IOInterface::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`
Syntax:	`IOInterface & operator= (const IOInterface &other)=delete;`
Description:	Copy-assign another IOInterface to this instance.

⌋

[SWS_CRYPT_30203] Definition of API function ara::crypto::IOInterface::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`
Syntax:	`IOInterface & operator= (IOInterface &&other)=delete;`
Description:	Move-assign another IOInterface to this instance.

⌋

[SWS_CRYPT_40995] Definition of API function ara::crypto::IOInterface::IOInterface

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`
Syntax:	`IOInterface (const IOInterface &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_40996] Definition of API function ara::crypto::IOInterface::IOInterface

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/io_interface.h"
Scope:	`class ara::crypto::IOInterface`
Syntax:	`IOInterface (IOInterface &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_10150] Definition of API function ara::crypto::operator==

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "equal" for `CryptoObjectUid` operands.	

⌋

[SWS_CRYPT_10151] Definition of API function ara::crypto::operator<

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "less than" for `CryptoObjectUid` operands.	

⌋

[SWS_CRYPT_10152] Definition of API function ara::crypto::operator>

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "greater than" for `CryptoObjectUid` operands.	

⌋

[SWS_CRYPT_10153] Definition of API function ara::crypto::operator!=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "not equal" for `CryptoObjectUid` operands.	

⌋

[SWS_CRYPT_10154] Definition of API function ara::crypto::operator<=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "less than or equal" for `CryptoObjectUid` operands.	

⌋

[SWS_CRYPT_10155] Definition of API function ara::crypto::operator>=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "greater than or equal" for `CryptoObjectUid` operands.	

⌋

[SWS_CRYPT_10451] Definition of API function ara::crypto::operator==

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/uuid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "equal" for `Uuid` operands.	

⌋

[SWS_CRYPT_10452] Definition of API function ara::crypto::operator<

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/uuid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "less than" for `Uuid` operands.	

⌋

[SWS_CRYPT_10453] Definition of API function ara::crypto::operator>

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/uuid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "greater than" for `Uuid` operands.	

⌋

[SWS_CRYPT_10454] Definition of API function ara::crypto::operator!=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/uuid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "not equal" for `Uuid` operands.	

⌋

[SWS_CRYPT_10455] Definition of API function ara::crypto::operator<=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/uuid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "less than or equal" for `Uuid` operands.	

⌋

[SWS_CRYPT_10456] Definition of API function ara::crypto::operator>=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/uuid.h"	
Scope:	`namespace ara::crypto`	
Syntax:	`constexpr bool 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:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator "greater than or equal" for `Uuid` operands.	

⌋

[SWS_CRYPT_19954] Definition of API function ara::crypto::CryptoErrorDomain::ThrowAsException

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_error_domain.h"	
Scope:	`class ara::crypto::CryptoErrorDomain`	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const override;`	
Parameters (in):	errorCode	an error code identifier from the CryptoErrc enumeration
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	throws exception of error code. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

[SWS_CRYPT_19902] Definition of API function ara::crypto::CryptoErrorDomain::CryptoErrorDomain

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	function
Header file:	#include "ara/crypto/common/crypto_error_domain.h"
Scope:	`class ara::crypto::CryptoErrorDomain`
Syntax:	`constexpr CryptoErrorDomain () noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Ctor of the CryptoErrorDomain.

⌋

[SWS_CRYPT_19950] Definition of API function ara::crypto::CryptoErrorDomain::Name

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_error_domain.h"	
Scope:	`class ara::crypto::CryptoErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Crypto" text
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	returns Text "Crypto"	

⌋

[SWS_CRYPT_19953] Definition of API function ara::crypto::CryptoErrorDomain::Message

Status: DRAFT
Upstream requirements: RS_CRYPTO_02310

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/crypto_error_domain.h"	
Scope:	`class ara::crypto::CryptoErrorDomain`	
Syntax:	`const char * 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:	exception safe	
Thread Safety:	implementation defined	
Description:	Translate an error code value into a text message.	

⌋

[SWS_CRYPT_10710] Definition of API function ara::crypto::Serializable::~Serializable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004, RS_CRYPTO_02302

⌈

Kind:	function
Header file:	#include "ara/crypto/common/serializable.h"
Scope:	`class ara::crypto::Serializable`
Syntax:	`virtual ~Serializable () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_10711] Definition of API function ara::crypto::Serializable::ExportPublicly

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/serializable.h"	
Scope:	`class ara::crypto::Serializable`	
Syntax:	`virtual ara::core::Result< std::size_t > ExportPublicly (ReadWriteMemRegion out, ara::core::Optional< FormatId > formatId) const noexcept=0;`	
Parameters (in):	formatId	the output format
Parameters (out):	out	Output buffer (span) to hold the serialized data
Return value:	ara::core::Result< std::size_t >	the number of Bytes written to the output buffer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnknownIdentifier	--
		if an unknown format ID was specified

▽

△

	ara::crypto::CryptoErrc::kUnsupportedFormat	--	
		if the specified format ID is not supported for this object type	
	ara::crypto::CryptoErrc::kInsufficientCapacity	--	
		if out does not have sufficient capacity	
Description:	This interface shall serialize the data of this object to the output buffer provided. If a format is not specified, the default export format of the CryptoProvider that holds this object shall be used.		

⌋

[SWS_CRYPT_41026] Definition of API function ara::crypto::Serializable::GetSerializedSize

Status: DRAFT
Upstream requirements: RS_CRYPTO_02112

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/serializable.h"	
Scope:	`class ara::crypto::Serializable`	
Syntax:	`virtual ara::core::Result< std::size_t > GetSerializedSize (FormatId formatId) const noexcept=0;`	
Parameters (in):	formatId	the output format
Return value:	ara::core::Result< std::size_t >	the number of Bytes required to hold the serialized data
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::crypto::CryptoErrc::kUnsupportedFormat	--
		if the specified format ID is not supported for this object type
Description:	This interface shall return the size of the output buffer required to hold the serialized data of this object in the requested format.	

⌋

[SWS_CRYPT_30204] Definition of API function ara::crypto::Serializable::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/serializable.h"
Scope:	`class ara::crypto::Serializable`
Syntax:	`Serializable & operator= (const Serializable &other)=delete;`
Description:	Copy-assign another Serializable to this instance.

⌋

[SWS_CRYPT_30205] Definition of API function ara::crypto::Serializable::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/serializable.h"
Scope:	`class ara::crypto::Serializable`
Syntax:	`Serializable & operator= (Serializable &&other)=delete;`
Description:	Move-assign another Serializable to this instance.

⌋

[SWS_CRYPT_40997] Definition of API function ara::crypto::Serializable::Serializable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/serializable.h"
Scope:	`class ara::crypto::Serializable`
Syntax:	`Serializable (const Serializable &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_40998] Definition of API function ara::crypto::Serializable::Serializable

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/serializable.h"
Scope:	`class ara::crypto::Serializable`
Syntax:	`Serializable (Serializable &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_10851] Definition of API function ara::crypto::VolatileTrustedContainer::~VolatileTrustedContainer

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Scope:	`class ara::crypto::VolatileTrustedContainer`
Syntax:	`virtual ~VolatileTrustedContainer () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor.

⌋

[SWS_CRYPT_10853] Definition of API function ara::crypto::VolatileTrustedContainer::GetIOInterface

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"	
Scope:	`class ara::crypto::VolatileTrustedContainer`	
Syntax:	`virtual IOInterface & GetIOInterface () const noexcept=0;`	
Return value:	IOInterface &	a reference to the IOInterface of this container
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Retrieve the IOInterface used for importing/exporting objects into this container.	

⌋

[SWS_CRYPT_30206] Definition of API function ara::crypto::VolatileTrustedContainer::operator=

Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Scope:	`class ara::crypto::VolatileTrustedContainer`
Syntax:	`VolatileTrustedContainer & operator= (const VolatileTrustedContainer &other)=delete;`
Description:	Copy-assign another VolatileTrustedContainer to this instance.

⌋

[SWS_CRYPT_30207] Definition of API function ara::crypto::VolatileTrustedContainer::operator=
Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Scope:	`class ara::crypto::VolatileTrustedContainer`
Syntax:	`VolatileTrustedContainer & operator= (VolatileTrustedContainer &&other)=delete;`
Description:	Move-assign another VolatileTrustedContainer to this instance.

⌋

[SWS_CRYPT_40999] Definition of API function ara::crypto::VolatileTrustedContainer::VolatileTrustedContainer
Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Scope:	`class ara::crypto::VolatileTrustedContainer`
Syntax:	`VolatileTrustedContainer (const VolatileTrustedContainer &)=delete;`
Description:	Copy-Constructor.

⌋

[SWS_CRYPT_41000] Definition of API function ara::crypto::VolatileTrustedContainer::VolatileTrustedContainer
Status: DRAFT
Upstream requirements: RS_CRYPTO_02004

⌈

Kind:	function
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Scope:	`class ara::crypto::VolatileTrustedContainer`
Syntax:	`VolatileTrustedContainer (VolatileTrustedContainer &&)=delete;`
Description:	Move-Constructor.

⌋

[SWS_CRYPT_10411] Definition of API function ara::crypto::Uuid::IsNil

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	function	
Header file:	#include "ara/crypto/common/uuid.h"	
Scope:	`struct ara::crypto::Uuid`	
Syntax:	`bool IsNil () const noexcept;`	
Return value:	bool	true if this identifier is "Nil" and false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether this identifier is the "Nil UUID" (according to RFC4122).	

⌋

[SWS_CRYPT_13102] Definition of API variable ara::crypto::kAllowDataDecryption

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDataDecryption
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDataDecryption {0x0002};`
Description:	The key/seed can be used for data decryption initialization (applicable to symmetric and asymmetric algorithms).

⌋

[SWS_CRYPT_13101] Definition of API variable ara::crypto::kAllowDataEncryption

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDataEncryption
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDataEncryption {0x0001};`

▽

△

Description:	The key/seed can be used for data encryption initialization (applicable to symmetric and asymmetric algorithms).

⌋

[SWS_CRYPT_13113] Definition of API variable ara::crypto::kAllowDerivedData Decryption

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedDataDecryption
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedDataDecryption {kAllowData Decryption << 16};`
Description:	A derived seed or symmetric key can be used for data decryption.

⌋

[SWS_CRYPT_13112] Definition of API variable ara::crypto::kAllowDerivedData Encryption

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedDataEncryption
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedDataEncryption {kAllowData Encryption << 16};`
Description:	A derived seed or symmetric key can be used for data encryption.

⌋

[SWS_CRYPT_13117] Definition of API variable ara::crypto::kAllowDerivedRngInit

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedRngInit
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedRngInit {kAllowRngInit << 16};`
Description:	A derived seed or symmetric key can be used for seeding of a RandomGeneratorContext.

⌋

[SWS_CRYPT_13121] Definition of API variable ara::crypto::kAllowDerivedExactModeOnly

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedExactModeOnly
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedExactModeOnly {kAllowExactModeOnly << 16};`
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`.

⌋

[SWS_CRYPT_13118] Definition of API variable ara::crypto::kAllowDerivedKdfMaterial

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedKdfMaterial
Type:	`const AllowedUsageFlags`

▽

△

Syntax:	`const AllowedUsageFlags kAllowDerivedKdfMaterial {kAllowKdfMaterial << 16};`
Description:	A derived seed or symmetric key can be used as a `RestrictedUseObject` for slave-keys derivation via a Key Derivation Function (KDF).

⌋

[SWS_CRYPT_13122] Definition of API variable ara::crypto::kAllowKdfMaterial AnyUsage

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowKdfMaterialAnyUsage
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowKdfMaterialAnyUsage { kAllowKdfMaterial | kAllowDerivedDataEncryption | kAllowDerivedDataDecryption | kAllowDerivedSignature | kAllowDerivedVerification | kAllowDerivedKeyDiversify | kAllowDerivedRngInit | kAllowDerivedKdfMaterial | kAllowDerivedKeyExporting | kAllowDerivedKeyImporting};`
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.

⌋

[SWS_CRYPT_13116] Definition of API variable ara::crypto::kAllowDerivedKey Diversify

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedKeyDiversify
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedKeyDiversify {kAllowKeyDiversify << 16};`
Description:	A derived seed or symmetric key can be used for slave-keys diversification.

⌋

[SWS_CRYPT_13119] Definition of API variable ara::crypto::kAllowDerivedKeyExporting

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedKeyExporting
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedKeyExporting {kAllowKeyExporting << 16};`
Description:	A derived seed or symmetric key can be used as a "transport" one for Key-Wrap transformation.

⌋

[SWS_CRYPT_13120] Definition of API variable ara::crypto::kAllowDerivedKeyImporting

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedKeyImporting
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedKeyImporting {kAllowKeyImporting << 16};`
Description:	A derived seed or symmetric key can be used as a "transport" one for Key-Unwrap transformation.

⌋

[SWS_CRYPT_13114] Definition of API variable ara::crypto::kAllowDerivedSignature

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedSignature

▽

△

Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedSignature {kAllowSignature << 16};`
Description:	A derived seed or symmetric key can be used for MAC/HMAC production.

⌋

[SWS_CRYPT_13115] Definition of API variable ara::crypto::kAllowDerivedVerification

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowDerivedVerification
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowDerivedVerification {kAllowVerification << 16};`
Description:	A derived seed or symmetric key can be used for MAC/HMAC verification.

⌋

[SWS_CRYPT_13111] Definition of API variable ara::crypto::kAllowExactModeOnly

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowExactModeOnly
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowExactModeOnly {0x8000};`
Description:	The key can be used only for the mode directly specified by `Key::AlgId`.

⌋

[SWS_CRYPT_13108] Definition of API variable ara::crypto::kAllowKdfMaterial

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowKdfMaterial
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowKdfMaterial {0x0080};`
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).

⌋

[SWS_CRYPT_13105] Definition of API variable ara::crypto::kAllowKeyAgreement

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowKeyAgreement
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowKeyAgreement {0x0010};`
Description:	The seed or asymmetric key can be used for key-agreement protocol execution.

⌋

[SWS_CRYPT_13106] Definition of API variable ara::crypto::kAllowKeyDiversify

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowKeyDiversify
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowKeyDiversify {0x0020};`
Description:	The seed or symmetric key can be used for slave-keys diversification.

⌋

[SWS_CRYPT_13109] Definition of API variable ara::crypto::kAllowKeyExporting

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowKeyExporting
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowKeyExporting {0x0100};`
Description:	The key can be used as "transport" one for Key-Wrap or Encapsulate transformations (applicable to symmetric and asymmetric keys).

⌋

[SWS_CRYPT_13110] Definition of API variable ara::crypto::kAllowKeyImporting

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowKeyImporting
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowKeyImporting {0x0200};`
Description:	The key can be used as "transport" one for Key-Unwrap or Decapsulate transformations (applicable to symmetric and asymmetric keys).

⌋

[SWS_CRYPT_40991] Definition of API variable ara::crypto::kAllowExport

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowExport
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowExport {0x0400};`
Description:	The key can be exported (if not set, export is not possible)

⌋

[SWS_CRYPT_41024] Definition of API variable ara::crypto::kAllowPersist

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowPersist
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowPersist = 0x0800;`
Description:	The key can be stored to a KeySlot.

⌋

[SWS_CRYPT_13100] Definition of API variable ara::crypto::kAllowPrototypedOnly

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowPrototypedOnly
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowPrototypedOnly {0};`
Description:	The key/seed usage will be fully specified by a key slot prototype (the object can be used only after reloading from the slot). This group contains list of constant 1-bit values predefined for Allowed Usage flags.

⌋

[SWS_CRYPT_13107] Definition of API variable ara::crypto::kAllowRngInit

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowRngInit
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowRngInit {0x0040};`

▽

△

Description:	The seed or symmetric key can be used for seeding of a RandomGeneratorCtx.

⌋

[SWS_CRYPT_13103] Definition of API variable ara::crypto::kAllowSignature

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowSignature
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowSignature {0x0004};`
Description:	The key/seed can be used for digital signature or MAC/HMAC production (applicable to symmetric and asymmetric algorithms).

⌋

[SWS_CRYPT_13104] Definition of API variable ara::crypto::kAllowVerification

Status: DRAFT
Upstream requirements: RS_CRYPTO_02111

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/base_id_types.h"
Scope:	`namespace ara::crypto`
Symbol:	kAllowVerification
Type:	`const AllowedUsageFlags`
Syntax:	`const AllowedUsageFlags kAllowVerification {0x0008};`
Description:	The key/seed can be used for digital signature or MAC/HMAC verification (applicable to symmetric and asymmetric algorithms).

⌋

[SWS_CRYPT_10102] Definition of API variable ara::crypto::CryptoObjectUid::mVersionStamp

Status: DRAFT
Upstream requirements: RS_CRYPTO_02006

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Scope:	`struct ara::crypto::CryptoObjectUid`
Symbol:	mVersionStamp
Type:	`std::uint64_t`
Syntax:	`std::uint64_t mVersionStamp = 0u;`
Description:	Sequential value of a steady timer or simple counter, representing version of correspondent Crypto Object.

⌋

[SWS_CRYPT_30002] Definition of API variable ara::crypto::SecureCounter::mLSQW

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/entry_point.h"
Scope:	`struct ara::crypto::SecureCounter`
Symbol:	mLSQW
Type:	`std::uint64_t`
Syntax:	`std::uint64_t mLSQW;`
Description:	least significant 64 bits

⌋

[SWS_CRYPT_30003] Definition of API variable ara::crypto::SecureCounter::mMSQW

Status: DRAFT
Upstream requirements: RS_CRYPTO_02401

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/entry_point.h"
Scope:	`struct ara::crypto::SecureCounter`
Symbol:	mMSQW
Type:	`std::uint64_t`
Syntax:	`std::uint64_t mMSQW;`

▽

△

Description:	most significant 64 bits

⌋

[SWS_CRYPT_10412] Definition of API variable ara::crypto::Uuid::mQwordLs

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/uuid.h"
Scope:	`struct ara::crypto::Uuid`
Symbol:	mQwordLs
Type:	`std::uint64_t`
Syntax:	`std::uint64_t mQwordLs = 0u;`
Description:	Less significant QWORD.

⌋

[SWS_CRYPT_10413] Definition of API variable ara::crypto::Uuid::mQwordMs

Status: DRAFT
Upstream requirements: RS_CRYPTO_02005

⌈

Kind:	variable
Header file:	#include "ara/crypto/common/uuid.h"
Scope:	`struct ara::crypto::Uuid`
Symbol:	mQwordMs
Type:	`std::uint64_t`
Syntax:	`std::uint64_t mQwordMs = 0u;`
Description:	Most significant QWORD.

⌋

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.

10 Configuration

The configuration model of this functional cluster is defined in [55]. This chapter defines the default values for attributes and semantic constraints for elements specified in [55] that are part of the configuration model of this functional cluster.

10.1 Default Values

This functional cluster does not define any default values for attributes specified in [55].

10.2 Semantic Constraints

This section defines semantic constraints for elements specified in [55] that are part of the configuration model of this functional cluster.

[SWS_CRYPT_CONSTR_00001] Configurable Namespace for Cryptography

Status: DRAFT

⌈`CryptoInterface.namespace` shall never exist.⌋

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	**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.1: CryptoCertificate

Class	**CryptoCertificateInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			

Note	This meta-class provides the ability to define a PortInterface for a CryptoCertificate.
	
	Tags:
	atp.Status=candidate
	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***
isPrivate	Boolean	0..1	attr	This attribute controls the possibility to access the content of the CryptoCertificateSlot by Find() interfaces of the X509 Provider. **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). **Tags:** atp.Status=candidate

Table A.2: 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.3: CryptoCertificateToCryptoKeySlotMapping

Class	**CryptoCertificateToPortPrototypeMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class represents the ability to define a mapping between a CryptoCertificate on target-configuration level to a given PortPrototype that is typed by a CryptoCertificateInterface. **Tags:** atp.recommendedPackage=CryptoCertificateToPortPrototypeMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
crypto Certificate	CryptoCertificate	0..1	ref	This reference represents the mapped cryptoCertificate.
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.
writeAccess	Boolean	0..1	attr	This attribute defines whether the application has write-access to the CryptoCertificate (true) or only read-access (false).

Table A.4: CryptoCertificateToPortPrototypeMapping

Class	***CryptoInterface*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign			
Note	This meta-class provides the abstract ability to define a PortInterface for the support of crypto use cases. **Tags:** atp.Status=candidate			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Subclasses	*AbstractCryptoKeySlotInterface*, CryptoCertificateInterface, CryptoProviderInterface, CryptoTrustMaster Interface			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.5: CryptoInterface

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. **Tags:** atp.ManifestKind=MachineManifest			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	CryptoProvider.keySlot			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allocateShadowCopy	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. 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.
cryptoKeySlotDesign	CryptoKeySlotDesign	0..1	ref	This reference identifies the CryptoKeySlotDesign from which the referencing CryptoKeySlot was derived.
cryptoObjectType	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. **Tags:** atp.Status=candidate
keySlotAllowedModification	CryptoKeySlotAllowedModification	0..1	aggr	Restricts how this keySlot may be used **Tags:** atp.Status=candidate
keySlotContentAllowedUsage	CryptoKeySlotContentAllowedUsage	*	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 cryptoObjectType and cryptoAlgId.
slotType	CryptoKeySlotTypeEnum	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: CryptoKeySlot

Class	**CryptoKeySlotInterface**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface

Note	This meta-class provides the ability to define a PortInterface for **using and modifying** Crypto Key Slots.
	
	Tags:
	atp.Status=candidate
	atp.recommendedPackage=CryptoInterfaces

▽

△

Class	**CryptoKeySlotInterface**			
Base	*ARElement, ARObject, AbstractCryptoKeySlotInterface, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, CryptoInterface, Identifiable, MultilanguageReferrable, Packageable Element, PortInterface, Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.7: CryptoKeySlotInterface

Class	**CryptoKeySlotToClientPortPrototypeMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment::Instance RefUsage			
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 CryptoKeyClientSlotInterface (this means only read is supported). **Tags:** atp.recommendedPackage=CryptoKeySlotMappings			
Base	*ARElement, ARObject, AbstractCryptoKeySlotToPortPrototypeMapping, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDeployment Element, UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.8: CryptoKeySlotToClientPortPrototypeMapping

Class	**CryptoKeySlotToPortPrototypeMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class represents the ability to define a mapping between a CryptoKeySlot on target-configuration level to a given PortPrototype that is typed by a CryptoKeySlotInterface. **Tags:** atp.recommendedPackage=CryptoKeySlotMappings			
Base	*ARElement, ARObject, AbstractCryptoKeySlotToPortPrototypeMapping, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDeployment Element, UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.9: 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.

▽

△

Class	**CryptoProvider**			
keySlot	CryptoKeySlot	*	aggr	This aggregation represents the key slots that are allocated by the CryptoProvider. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=keySlot.shortName

Table A.10: CryptoProvider

Class	**CryptoProviderInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign			

Note	This meta-class provides the ability to define a PortInterface for a CryptoProvider.
	Tags:
	atp.Status=candidate
	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.11: 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. **Tags:** atp.recommendedPackage=CryptoProviderToPortPrototypeMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *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. **InstanceRef implemented by:** RPortPrototypeIn ExecutableInstanceRef
process	Process	0..1	ref	This reference represents the process required as context for the mapping.

Table A.12: CryptoProviderToPortPrototypeMapping

Class	**CryptoServiceCertificate**			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	This meta-class represents the ability to model a cryptographic certificate. **Tags:** atp.recommendedPackage=CryptoServiceCertificates			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	**CryptoServiceCertificate**			
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.
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.13: 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, *IdsmAbstractPort Interface*, LogAndTraceInterface, ModeSwitchInterface, NetworkManagementPortInterface, *Persistency Interface*, *PlatformHealthManagementInterface*, 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

Table A.14: 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*, *UploadableDeploymentElement*, *Uploadable PackageElement*			
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.

▽

△

Class	**Process**			
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
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. This attribute is used to support the modeling of execution dependencies that utilize the condition of process state. Please note that the process states may not be modeled arbitrarily at any stage of the AUTOSAR workflow because the supported states are standardized in the context of the SWS Execution Management [56].
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations.

Table A.15: 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.16: 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.

C Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

C.1 Traceable item history of this document according to AUTOSAR Release R24-11

C.1.1 Added Specification Items in R24-11

[SWS_CRYPT_41019] [SWS_CRYPT_41020] [SWS_CRYPT_41021] [SWS_-CRYPT_41022] [SWS_CRYPT_41023] [SWS_CRYPT_41024] [SWS_CRYPT_41025] [SWS_CRYPT_41026] [SWS_CRYPT_41027] [SWS_CRYPT_41028] [SWS_-CRYPT_41029] [SWS_CRYPT_41030] [SWS_CRYPT_41031] [SWS_CRYPT_41032] [SWS_CRYPT_41033] [SWS_CRYPT_41034] [SWS_CRYPT_41035] [SWS_-CRYPT_41036] [SWS_CRYPT_41037] [SWS_CRYPT_41038] [SWS_CRYPT_41039] [SWS_CRYPT_41040] [SWS_CRYPT_41041] [SWS_CRYPT_41042] [SWS_-CRYPT_41043] [SWS_CRYPT_41044] [SWS_CRYPT_41045] [SWS_CRYPT_41046] [SWS_CRYPT_41047] [SWS_CRYPT_41048] [SWS_CRYPT_41049] [SWS_-CRYPT_41050] [SWS_CRYPT_41051] [SWS_CRYPT_41052] [SWS_CRYPT_41053] [SWS_CRYPT_41054] [SWS_CRYPT_41055] [SWS_CRYPT_41056] [SWS_-CRYPT_41057] [SWS_CRYPT_41058]

C.1.2 Changed Specification Items in R24-11

[SWS_CRYPT_00500] [SWS_CRYPT_00501] [SWS_CRYPT_00504] [SWS_-CRYPT_00505] [SWS_CRYPT_00506] [SWS_CRYPT_00611] [SWS_CRYPT_00906] [SWS_CRYPT_00907] [SWS_CRYPT_00919] [SWS_CRYPT_01201] [SWS_-CRYPT_01210] [SWS_CRYPT_01503] [SWS_CRYPT_01504] [SWS_CRYPT_01653] [SWS_CRYPT_01654] [SWS_CRYPT_01655] [SWS_CRYPT_01656] [SWS_-CRYPT_01657] [SWS_CRYPT_01800] [SWS_CRYPT_01804] [SWS_CRYPT_01805] [SWS_CRYPT_01807] [SWS_CRYPT_01808] [SWS_CRYPT_01811] [SWS_-CRYPT_02105] [SWS_CRYPT_02109] [SWS_CRYPT_02415] [SWS_CRYPT_02416] [SWS_CRYPT_02417] [SWS_CRYPT_02419] [SWS_CRYPT_02420] [SWS_-CRYPT_02422] [SWS_CRYPT_02704] [SWS_CRYPT_02705] [SWS_CRYPT_02726] [SWS_CRYPT_03002] [SWS_CRYPT_03003] [SWS_CRYPT_04202] [SWS_-CRYPT_04203] [SWS_CRYPT_04204] [SWS_CRYPT_04213] [SWS_CRYPT_10003] [SWS_CRYPT_10016] [SWS_CRYPT_10099] [SWS_CRYPT_10111] [SWS_-CRYPT_10112] [SWS_CRYPT_10113] [SWS_CRYPT_10114] [SWS_CRYPT_10115] [SWS_CRYPT_10150] [SWS_CRYPT_10151] [SWS_CRYPT_10152] [SWS_-CRYPT_10153] [SWS_CRYPT_10154] [SWS_CRYPT_10155] [SWS_CRYPT_10411]

[SWS_CRYPT_10451] [SWS_CRYPT_10452] [SWS_CRYPT_10453] [SWS_-CRYPT_10454] [SWS_CRYPT_10455] [SWS_CRYPT_10456] [SWS_CRYPT_10701] [SWS_CRYPT_10710] [SWS_CRYPT_10711] [SWS_CRYPT_10808] [SWS_-CRYPT_10810] [SWS_CRYPT_10811] [SWS_CRYPT_10812] [SWS_CRYPT_10813] [SWS_CRYPT_10814] [SWS_CRYPT_10815] [SWS_CRYPT_10816] [SWS_-CRYPT_10817] [SWS_CRYPT_10818] [SWS_CRYPT_10819] [SWS_CRYPT_10821] [SWS_CRYPT_10822] [SWS_CRYPT_10823] [SWS_CRYPT_10851] [SWS_-CRYPT_10853] [SWS_CRYPT_19902] [SWS_CRYPT_19906] [SWS_CRYPT_19950] [SWS_CRYPT_19951] [SWS_CRYPT_19952] [SWS_CRYPT_19953] [SWS_-CRYPT_19954] [SWS_CRYPT_20098] [SWS_CRYPT_20099] [SWS_CRYPT_20102] [SWS_CRYPT_20103] [SWS_CRYPT_20312] [SWS_CRYPT_20313] [SWS_-CRYPT_20314] [SWS_CRYPT_20316] [SWS_CRYPT_20401] [SWS_CRYPT_20411] [SWS_CRYPT_20412] [SWS_CRYPT_20414] [SWS_CRYPT_20503] [SWS_-CRYPT_20505] [SWS_CRYPT_20512] [SWS_CRYPT_20513] [SWS_CRYPT_20514] [SWS_CRYPT_20515] [SWS_CRYPT_20516] [SWS_CRYPT_20517] [SWS_-CRYPT_20518] [SWS_CRYPT_20651] [SWS_CRYPT_20652] [SWS_CRYPT_20654] [SWS_CRYPT_20710] [SWS_CRYPT_20711] [SWS_CRYPT_20712] [SWS_-CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_20723] [SWS_CRYPT_20724] [SWS_CRYPT_20725] [SWS_CRYPT_20726] [SWS_CRYPT_20727] [SWS_-CRYPT_20728] [SWS_CRYPT_20729] [SWS_CRYPT_20730] [SWS_CRYPT_20731] [SWS_CRYPT_20732] [SWS_CRYPT_20733] [SWS_CRYPT_20741] [SWS_-CRYPT_20742] [SWS_CRYPT_20743] [SWS_CRYPT_20744] [SWS_CRYPT_20745] [SWS_CRYPT_20746] [SWS_CRYPT_20747] [SWS_CRYPT_20748] [SWS_-CRYPT_20750] [SWS_CRYPT_20751] [SWS_CRYPT_20752] [SWS_CRYPT_20753] [SWS_CRYPT_20754] [SWS_CRYPT_20755] [SWS_CRYPT_20756] [SWS_-CRYPT_20757] [SWS_CRYPT_20758] [SWS_CRYPT_20762] [SWS_CRYPT_20763] [SWS_CRYPT_20764] [SWS_CRYPT_20765] [SWS_CRYPT_20802] [SWS_-CRYPT_20810] [SWS_CRYPT_20811] [SWS_CRYPT_20812] [SWS_CRYPT_21002] [SWS_CRYPT_21010] [SWS_CRYPT_21011] [SWS_CRYPT_21012] [SWS_-CRYPT_21102] [SWS_CRYPT_21110] [SWS_CRYPT_21111] [SWS_CRYPT_21112] [SWS_CRYPT_21113] [SWS_CRYPT_21114] [SWS_CRYPT_21115] [SWS_-CRYPT_21116] [SWS_CRYPT_21118] [SWS_CRYPT_21302] [SWS_CRYPT_21311] [SWS_CRYPT_21312] [SWS_CRYPT_21313] [SWS_CRYPT_21314] [SWS_-CRYPT_21315] [SWS_CRYPT_21402] [SWS_CRYPT_21411] [SWS_CRYPT_21412] [SWS_CRYPT_21413] [SWS_CRYPT_21414] [SWS_CRYPT_21415] [SWS_-CRYPT_21416] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21514] [SWS_CRYPT_21515] [SWS_CRYPT_21516] [SWS_CRYPT_21517] [SWS_-CRYPT_21519] [SWS_CRYPT_21520] [SWS_CRYPT_21521] [SWS_CRYPT_21522] [SWS_CRYPT_21523] [SWS_CRYPT_21524] [SWS_CRYPT_21525] [SWS_-CRYPT_21715] [SWS_CRYPT_21802] [SWS_CRYPT_21810] [SWS_CRYPT_21813] [SWS_CRYPT_21815] [SWS_CRYPT_21816] [SWS_CRYPT_21817] [SWS_-CRYPT_21818] [SWS_CRYPT_22102] [SWS_CRYPT_22110] [SWS_CRYPT_22111] [SWS_CRYPT_22112] [SWS_CRYPT_22113] [SWS_CRYPT_22114] [SWS_-CRYPT_22115] [SWS_CRYPT_22116] [SWS_CRYPT_22118] [SWS_CRYPT_22120] [SWS_CRYPT_22210] [SWS_CRYPT_22211] [SWS_CRYPT_22212] [SWS_-CRYPT_22213] [SWS_CRYPT_22214] [SWS_CRYPT_22215] [SWS_CRYPT_22511]

[SWS_CRYPT_22711] [SWS_CRYPT_22712] [SWS_CRYPT_22902] [SWS_-CRYPT_22911] [SWS_CRYPT_22912] [SWS_CRYPT_22913] [SWS_CRYPT_22914] [SWS_CRYPT_22915] [SWS_CRYPT_23011] [SWS_CRYPT_23012] [SWS_-CRYPT_23013] [SWS_CRYPT_23014] [SWS_CRYPT_23015] [SWS_CRYPT_23016] [SWS_CRYPT_23210] [SWS_CRYPT_23211] [SWS_CRYPT_23212] [SWS_-CRYPT_23213] [SWS_CRYPT_23214] [SWS_CRYPT_23215] [SWS_CRYPT_23510] [SWS_CRYPT_23511] [SWS_CRYPT_23512] [SWS_CRYPT_23513] [SWS_-CRYPT_23515] [SWS_CRYPT_23516] [SWS_CRYPT_23602] [SWS_CRYPT_23611] [SWS_CRYPT_23612] [SWS_CRYPT_23613] [SWS_CRYPT_23614] [SWS_-CRYPT_23615] [SWS_CRYPT_23616] [SWS_CRYPT_23618] [SWS_CRYPT_23620] [SWS_CRYPT_23621] [SWS_CRYPT_23622] [SWS_CRYPT_23623] [SWS_-CRYPT_23624] [SWS_CRYPT_23625] [SWS_CRYPT_23626] [SWS_CRYPT_23627] [SWS_CRYPT_23634] [SWS_CRYPT_23635] [SWS_CRYPT_23702] [SWS_-CRYPT_23710] [SWS_CRYPT_23711] [SWS_CRYPT_23712] [SWS_CRYPT_23715] [SWS_CRYPT_23716] [SWS_CRYPT_23911] [SWS_CRYPT_24002] [SWS_-CRYPT_24011] [SWS_CRYPT_24012] [SWS_CRYPT_24013] [SWS_CRYPT_24014] [SWS_CRYPT_24015] [SWS_CRYPT_24016] [SWS_CRYPT_24018] [SWS_-CRYPT_24019] [SWS_CRYPT_24102] [SWS_CRYPT_24111] [SWS_CRYPT_24112] [SWS_CRYPT_24114] [SWS_CRYPT_24115] [SWS_CRYPT_24116] [SWS_-CRYPT_24410] [SWS_CRYPT_24411] [SWS_CRYPT_24412] [SWS_CRYPT_24413] [SWS_CRYPT_24414] [SWS_CRYPT_24415] [SWS_CRYPT_24714] [SWS_-CRYPT_24715] [SWS_CRYPT_24811] [SWS_CRYPT_29002] [SWS_CRYPT_29003] [SWS_CRYPT_29004] [SWS_CRYPT_29012] [SWS_CRYPT_29013] [SWS_-CRYPT_29014] [SWS_CRYPT_29015] [SWS_CRYPT_29021] [SWS_CRYPT_29022] [SWS_CRYPT_29023] [SWS_CRYPT_29032] [SWS_CRYPT_29033] [SWS_-CRYPT_29034] [SWS_CRYPT_29035] [SWS_CRYPT_29041] [SWS_CRYPT_29043] [SWS_CRYPT_29044] [SWS_CRYPT_29045] [SWS_CRYPT_29046] [SWS_-CRYPT_29047] [SWS_CRYPT_29048] [SWS_CRYPT_29049] [SWS_CRYPT_30098] [SWS_CRYPT_30099] [SWS_CRYPT_30110] [SWS_CRYPT_30115] [SWS_-CRYPT_30123] [SWS_CRYPT_30124] [SWS_CRYPT_30125] [SWS_CRYPT_30130] [SWS_CRYPT_30202] [SWS_CRYPT_30203] [SWS_CRYPT_30204] [SWS_-CRYPT_30205] [SWS_CRYPT_30206] [SWS_CRYPT_30207] [SWS_CRYPT_30208] [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_30301] [SWS_CRYPT_30306] [SWS_CRYPT_30350] [SWS_CRYPT_30351] [SWS_CRYPT_30401] [SWS_-CRYPT_30403] [SWS_CRYPT_30404] [SWS_CRYPT_30405] [SWS_CRYPT_30406] [SWS_CRYPT_30407] [SWS_CRYPT_30408] [SWS_CRYPT_30409] [SWS_-CRYPT_30510] [SWS_CRYPT_30550] [SWS_CRYPT_30551] [SWS_CRYPT_40099] [SWS_CRYPT_40111] [SWS_CRYPT_40112] [SWS_CRYPT_40113] [SWS_-CRYPT_40114] [SWS_CRYPT_40115] [SWS_CRYPT_40211] [SWS_CRYPT_40214] [SWS_CRYPT_40215] [SWS_CRYPT_40216] [SWS_CRYPT_40217] [SWS_-CRYPT_40218] [SWS_CRYPT_40220] [SWS_CRYPT_40311] [SWS_CRYPT_40313]

[SWS_CRYPT_40314] [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_40511] [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_40626] [SWS_-CRYPT_40627] [SWS_CRYPT_40628] [SWS_CRYPT_40629] [SWS_CRYPT_40630] [SWS_CRYPT_40631] [SWS_CRYPT_40632] [SWS_CRYPT_40633] [SWS_-CRYPT_40634] [SWS_CRYPT_40635] [SWS_CRYPT_40636] [SWS_CRYPT_40640] [SWS_CRYPT_40641] [SWS_CRYPT_40711] [SWS_CRYPT_40811] [SWS_-CRYPT_40911] [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_40951] [SWS_CRYPT_40970] [SWS_-CRYPT_40981] [SWS_CRYPT_40983] [SWS_CRYPT_40995] [SWS_CRYPT_40996] [SWS_CRYPT_40997] [SWS_CRYPT_40998] [SWS_CRYPT_40999] [SWS_-CRYPT_41000] [SWS_CRYPT_41001] [SWS_CRYPT_41002] [SWS_CRYPT_41003] [SWS_CRYPT_41004] [SWS_CRYPT_41005] [SWS_CRYPT_41006] [SWS_-CRYPT_41007] [SWS_CRYPT_41008] [SWS_CRYPT_41009] [SWS_CRYPT_41010] [SWS_CRYPT_41011] [SWS_CRYPT_41012] [SWS_CRYPT_41013] [SWS_-CRYPT_41014] [SWS_CRYPT_41015] [SWS_CRYPT_41016] [SWS_CRYPT_41017] [SWS_CRYPT_41018]

C.1.3 Deleted Specification Items in R24-11

[SWS_CRYPT_00104] [SWS_CRYPT_01208] [SWS_CRYPT_01209] [SWS_-CRYPT_01213] [SWS_CRYPT_02421] [SWS_CRYPT_02706] [SWS_CRYPT_10005] [SWS_CRYPT_10750] [SWS_CRYPT_10751] [SWS_CRYPT_10752] [SWS_-CRYPT_10753] [SWS_CRYPT_13000] [SWS_CRYPT_13001] [SWS_CRYPT_13002] [SWS_CRYPT_13003] [SWS_CRYPT_20319] [SWS_CRYPT_20760] [SWS_-CRYPT_20761] [SWS_CRYPT_22119] [SWS_CRYPT_23300] [SWS_CRYPT_23301] [SWS_CRYPT_23302] [SWS_CRYPT_23311] [SWS_CRYPT_23312] [SWS_-CRYPT_24113] [SWS_CRYPT_30126] [SWS_CRYPT_30131] [SWS_CRYPT_40315] [SWS_CRYPT_40960] [SWS_CRYPT_40961] [SWS_CRYPT_40986] [SWS_-CRYPT_40987]

C.1.4 Added Constraints in R24-11

[SWS_CRYPT_CONSTR_00001]

C.1.5 Changed Constraints in R24-11

none

C.1.6 Deleted Constraints in R24-11

none

C.2 Traceable item history of this document according to AUTOSAR Release R23-11

C.2.1 Added Specification Items in R23-11

[SWS_CRYPT_19952] [SWS_CRYPT_40986] [SWS_CRYPT_40987] [SWS_CRYPT_40988] [SWS_CRYPT_40989] [SWS_CRYPT_40990] [SWS_CRYPT_40991] [SWS_CRYPT_40992] [SWS_CRYPT_40993] [SWS_CRYPT_40994] [SWS_CRYPT_40995] [SWS_CRYPT_40996] [SWS_CRYPT_40997] [SWS_CRYPT_40998] [SWS_CRYPT_40999] [SWS_CRYPT_41000] [SWS_CRYPT_41001] [SWS_CRYPT_41002] [SWS_CRYPT_41003] [SWS_CRYPT_41004] [SWS_CRYPT_41005] [SWS_CRYPT_41006] [SWS_CRYPT_41007] [SWS_CRYPT_41008] [SWS_CRYPT_41009] [SWS_CRYPT_41010] [SWS_CRYPT_41011] [SWS_CRYPT_41012] [SWS_CRYPT_41013] [SWS_CRYPT_41014] [SWS_CRYPT_41015] [SWS_CRYPT_41016] [SWS_CRYPT_41017] [SWS_CRYPT_41018]

C.2.2 Changed Specification Items in R23-11

[SWS_CRYPT_01207] [SWS_CRYPT_01211] [SWS_CRYPT_01502] [SWS_CRYPT_01503] [SWS_CRYPT_01659] [SWS_CRYPT_01807] [SWS_CRYPT_02122] [SWS_CRYPT_04204] [SWS_CRYPT_10000] [SWS_CRYPT_10003] [SWS_CRYPT_10005] [SWS_CRYPT_10099] [SWS_CRYPT_10114] [SWS_CRYPT_10305] [SWS_CRYPT_10306] [SWS_CRYPT_10811] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_20723] [SWS_CRYPT_20730] [SWS_CRYPT_21115] [SWS_CRYPT_21116] [SWS_CRYPT_21519] [SWS_CRYPT_21523] [SWS_CRYPT_22115] [SWS_CRYPT_22116] [SWS_CRYPT_22118] [SWS_CRYPT_22215] [SWS_CRYPT_22911] [SWS_CRYPT_22912] [SWS_CRYPT_22913] [SWS_CRYPT_22914] [SWS_CRYPT_23215] [SWS_CRYPT_23618] [SWS_CRYPT_23623] [SWS_CRYPT_23710] [SWS_CRYPT_23911] [SWS_CRYPT_24018] [SWS_CRYPT_30099] [SWS_CRYPT_30202] [SWS_CRYPT_30203] [SWS_CRYPT_30204] [SWS_CRYPT_30205] [SWS_CRYPT_30206] [SWS_CRYPT_30207] [SWS_CRYPT_30208] [SWS_CRYPT_30209] [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_40099] [SWS_CRYPT_40203] [SWS_CRYPT_40216] [SWS_-CRYPT_40217] [SWS_CRYPT_40218] [SWS_CRYPT_40600] [SWS_CRYPT_40614] [SWS_CRYPT_40616] [SWS_CRYPT_40617] [SWS_CRYPT_40618] [SWS_-CRYPT_40619] [SWS_CRYPT_40620] [SWS_CRYPT_40621] [SWS_CRYPT_40622] [SWS_CRYPT_40628] [SWS_CRYPT_40629] [SWS_CRYPT_40630] [SWS_-CRYPT_40944] [SWS_CRYPT_40945] [SWS_CRYPT_40962]

C.2.3 Deleted Specification Items in R23-11

[SWS_CRYPT_00622] [SWS_CRYPT_10042] [SWS_CRYPT_10300] [SWS_-CRYPT_10301] [SWS_CRYPT_10303] [SWS_CRYPT_10304] [SWS_CRYPT_10712] [SWS_CRYPT_20813] [SWS_CRYPT_21013] [SWS_CRYPT_21117] [SWS_-CRYPT_22117] [SWS_CRYPT_22216] [SWS_CRYPT_22713] [SWS_CRYPT_23216] [SWS_CRYPT_23514] [SWS_CRYPT_23617] [SWS_CRYPT_23619] [SWS_-CRYPT_23717] [SWS_CRYPT_24017] [SWS_CRYPT_40212] [SWS_CRYPT_40219] [SWS_CRYPT_40221] [SWS_CRYPT_40602] [SWS_CRYPT_40603] [SWS_-CRYPT_40624] [SWS_CRYPT_40625] [SWS_CRYPT_40637] [SWS_CRYPT_40638] [SWS_CRYPT_40639]