Document Title	Specification of Cryptography for Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	883

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R20-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
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	19-03	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	18-10	AUTOSAR Release Management	• Removed crypto API introduced in release 17-10
2018-03-29	18-03	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	17-10	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview 5

2 Acronyms and Abbreviations 6

3 Related documentation 10

3.1 Input documents & related standards and norms 10
3.2 Further applicable specification 13

4 Constraints and assumptions 14

4.1 Constraints 14
4.2 Assumptions 14
4.3 Known limitations 14
4.4 Applicability to car domains 15

5 Dependencies to other functional clusters 16

5.1 Protocol layer dependencies 16

6 Requirements Tracing 17

7 Functional specification 39

7.1 Functional Cluster Lifecycle 39
7.1.1 Startup 39
7.1.2 Shutdown 39
7.2 Architectural concepts 40
7.2.1 Integration with Identity and Access Management 42
7.2.2 Integration into AUTOSAR 43
7.2.3 Application level 45
7.2.4 System service level 46
7.2.5 Bridging domains: the IOInterface 46
7.3 Crypto API structure 47
7.4 Crypto API elements 48
7.4.1 Crypto Provider 48
7.4.1.1 Random Number Generator (RNG) 49
7.4.1.2 Key Derivation Function (KDF) 50
7.4.1.3 Hashing 52
7.4.1.4 Message Authentication Code (MAC) 54
7.4.1.5 Symmetric encryption 57
7.4.1.6 Authenticated Encryption 60
7.4.1.7 Key Wrapping 61
7.4.1.8 Digital signatures 62
7.4.1.9 Asymmetric encryption 66
7.4.1.10 Key Encapsulation Mechanism (KEM) 68
7.4.1.11 Key Exchange Protocol, Key Exchange Mechanism, and Key Exchange Scheme 69
7.4.1.12 Identification of cryptographic primitives and using one 72

7.4.1.13 Support on internal elements (Loading, Update, Import, and Export) . . . 73
7.4.2 Key Storage Provider . . . 74
7.4.2.1 Serializable interface . . . 76
7.4.2.2 Key Generation . . . 77
7.4.2.3 Exporting and Importing of Key Material . . . 78
7.4.3 Certificate handling (X.509 Provider) . . . 78
7.4.3.1 Certificate Signing Request . . . 82
7.4.3.2 Using Certificates . . . 82
7.4.3.3 Revocation of certificates . . . 86
7.5 Cryptographic Primitives Naming Convention . . . 87

8 API specification 91

8.1 C++ language binding Crypto Provider . . . 91
8.2 C++ language binding Key Storage Provider . . . 224
8.3 C++ language binding X509 Certificate Management Provider . . . 243
8.4 API Common Data Types . . . 283
8.5 API Reference . . . 291

9 Service Interfaces 322

9.1 Type definitions . . . 322
9.2 Provided Service Interfaces . . . 322
9.3 Required Service Interfaces . . . 322
9.4 Application Errors . . . 322

A Mentioned Manifest Elements 323

B Interfaces to other Functional Clusters (informative) 330

B.1 Overview . . . 330
B.2 Interface Tables . . . 330

C History of Constraints and Specification Items 331

C.1 Constraint and Specification Item History of this document according to AUTOSAR Release yy-mm . . . 331

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 Cluster 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
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
KSP	Key Storage Provider
MAC	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.
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.
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

Abbreviation / Acronym:	Description:
TLS	Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network.
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.
AsymmetricKey	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.
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.
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.
Key Storage Provider	A structural element that organizes and manages cryptographic keys.

Terms:	Description:
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.
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 plain text. A symmetric key is also called secret key because it must be kept secret.
X.509 Provider	Domain SW for X.509 certificates parsing, verification, storage and search.

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

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

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

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

[5] Specification of the Adaptive Core
AUTOSAR_SWS_AdaptiveCore

[6] Requirements on Security Management for Adaptive Platform
AUTOSAR_RS_SecurityManagement

[7] 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]

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

[9] 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

[10] NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions
http://csrc.nist.gov/publications/drafts/800-108/Draft_SP-800-108_April-2008.pdf

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

[12] ISO IEC 9797-3:2011 Amd 1:2020(en) Information technology - Security techniques - Message Authentication Codes (MAC)

http://www.iso.org

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

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

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

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

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

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

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

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

[21] TRIVIUM Specifications

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

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

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

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

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

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

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

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

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

[31] X.509

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Further applicable specification

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

4 Constraints and assumptions

4.1 Constraints

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

- Support the independence of application software components from a specific platform implementation.
- Make the API as lean as possible, no specific use cases are supported, which could also be layered on top of the API.
- Offer a ”comfort layer” to enable the use of C++11/14 features.
- Support the integration into safety relevant systems.
- Support the integration into cyber security relevant systems.

4.2 Assumptions

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

Beside the support of applications and functional clusters, the FC Crypto provides mechanism to ensure secure communication. The FC Crypto helps Adaptive application and functional cluster to establish secure channels. The FC Crypto also allows to store data persistent in an encrypted manner.

4.3 Known limitations

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

- **Asynchronous interfaces**
  Currently there is only a synchronous API specification and asynchronous behavior (if required) should be implemented on the consumer application level. It can be done via utilization of dedicated execution threads for long-time operations.
- **Full X.509 certificate support incl. OCSP and OCSP stabling**
  CryptoAPI doesn’t provide complete specification of the X.509 certificates man-

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

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

4.4 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other functional clusters

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

5.1 Protocol layer dependencies

There are currently no dependencies to protocol layers.

6 Requirements Tracing

The following tables reference the requirements specified in [6] and 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] [SWS_CRYPT_20760] [SWS_CRYPT_20761]
[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]

Requirement	Description	Satisfied by
[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]
[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_04206] [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_10750] [SWS_CRYPT_10751] [SWS_CRYPT_10752] [SWS_CRYPT_10753] [SWS_CRYPT_10800] [SWS_CRYPT_10810]

Requirement	Description	Satisfied by
		[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] [SWS_CRYPT_30123] [SWS_CRYPT_30124] [SWS_CRYPT_30125] [SWS_CRYPT_30126] [SWS_CRYPT_30200] [SWS_CRYPT_30201] [SWS_CRYPT_30202] [SWS_CRYPT_30203] [SWS_CRYPT_30204] [SWS_CRYPT_30205] [SWS_CRYPT_30206] [SWS_CRYPT_30207] [SWS_CRYPT_30209] [SWS_CRYPT_30210] [SWS_CRYPT_30211] [SWS_CRYPT_30212] [SWS_CRYPT_30213] [SWS_CRYPT_30214] [SWS_CRYPT_30215] [SWS_CRYPT_30216] [SWS_CRYPT_30217] [SWS_CRYPT_30218] [SWS_CRYPT_30219] [SWS_CRYPT_30220] [SWS_CRYPT_30221] [SWS_CRYPT_30222] [SWS_CRYPT_30223] [SWS_CRYPT_30224] [SWS_CRYPT_30225] [SWS_CRYPT_30226] [SWS_CRYPT_30227] [SWS_CRYPT_30404] [SWS_CRYPT_30406] [SWS_CRYPT_30408] [SWS_CRYPT_30409]

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

Requirement	Description	Satisfied by
[RS_CRYPTO_-02006]	The Crypto Stack shall support a version control mechanism and distinguish "versions" and "origin sources" of cryptographic objects	[SWS_CRYPT_04201] [SWS_CRYPT_04212] [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_20760] [SWS_CRYPT_20761] [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]
[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]

Requirement	Description	Satisfied by
[RS_CRYPTO_-02008]	The Crypto Stack shall support restrictions of the allowed usage scope for keys and "secret seeds"	[SWS_CRYPT_10004] [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]
[RS_CRYPTO_-02100]	No description	[SWS_CRYPT_03300]
[RS_CRYPTO_-02101]	The Crypto Stack shall provide interfaces to generate cryptographic keys for all supported primitives	[SWS_CRYPT_00601] [SWS_CRYPT_00602] [SWS_CRYPT_00603] [SWS_CRYPT_00608] [SWS_CRYPT_00609] [SWS_CRYPT_00610] [SWS_CRYPT_00611] [SWS_CRYPT_00622] [SWS_CRYPT_03311] [SWS_CRYPT_10300] [SWS_CRYPT_10301] [SWS_CRYPT_10303] [SWS_CRYPT_10304] [SWS_CRYPT_20721] [SWS_CRYPT_20722]
[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_21510] [SWS_CRYPT_21510] [SWS_CRYPT_21510] [SWS_CRYPT_21513] [SWS_CRYPT_21515] [SWS_CRYPT_21523] [SWS_CRYPT_21813]

Requirement	Description	Satisfied by
[RS_CRYPTO_-02103]	The Crypto Stack shall support primitives to derive cryptographic key material from a base key material	[SWS_CRYPT_03313] [SWS_CRYPT_10402] [SWS_CRYPT_20748] [SWS_CRYPT_21500] [SWS_CRYPT_21501] [SWS_CRYPT_21518] [SWS_CRYPT_21518] [SWS_CRYPT_21520] [SWS_CRYPT_21522]
[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_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_03303]
[RS_CRYPTO_-02107]	The Crypto Stack shall support the algorithm specification in any key generation or derivation request	[SWS_CRYPT_01501] [SWS_CRYPT_03304] [SWS_CRYPT_10014] [SWS_CRYPT_13000] [SWS_CRYPT_13001] [SWS_CRYPT_13002] [SWS_CRYPT_13003] [SWS_CRYPT_20710] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21510] [SWS_CRYPT_21510] [SWS_CRYPT_21510] [SWS_CRYPT_21513] [SWS_CRYPT_21515] [SWS_CRYPT_21523]

Requirement	Description	Satisfied by
[RS_CRYPTO_-02108]	The Crypto Stack shall provide interfaces for management and usage of algorithm-specific domain parameters	[SWS_CRYPT_03305] [SWS_CRYPT_20414] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21314] [SWS_CRYPT_21412] [SWS_CRYPT_21414] [SWS_CRYPT_21510] [SWS_CRYPT_21510] [SWS_CRYPT_21510] [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_23714] [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_03306] [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]

Requirement	Description	Satisfied by
[RS_CRYPTO_02110]	The Crypto Stack shall support prototyping of application-exclusive key slot resources	[SWS_CRYPT_00101] [SWS_CRYPT_03307] [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]
[RS_CRYPTO_02111]	The Crypto Stack shall provide applications a possibility to define usage restrictions of any new generated or derived key	[SWS_CRYPT_03308] [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]

Requirement	Description	Satisfied by
		[SWS_CRYPT_13122] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21510] [SWS_CRYPT_21510] [SWS_CRYPT_21510] [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]
[RS_CRYPTO_-02112]	The Crypto Stack shall execute export/import of a key value together with its meta information	[SWS_CRYPT_03309] [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_10712] [SWS_CRYPT_20005] [SWS_CRYPT_20728] [SWS_CRYPT_20729] [SWS_CRYPT_20730] [SWS_CRYPT_20731] [SWS_CRYPT_20732]
[RS_CRYPTO_-02113]	The Crypto Stack interfaces shall support control of the exportability property of a key object	[SWS_CRYPT_03310] [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_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21312] [SWS_CRYPT_21412] [SWS_CRYPT_21515] [SWS_CRYPT_21523] [SWS_CRYPT_21813] [SWS_CRYPT_22511] [SWS_CRYPT_24016] [SWS_CRYPT_24017]
[RS_CRYPTO_-02116]	The Crypto Stack shall support version control of key objects kept in the Key Storage	[SWS_CRYPT_30300]

Requirement	Description	Satisfied by
[RS_CRYPTO_-02201]	The Crypto Stack shall provide interfaces to use symmetric encryption and decryption primitives	[SWS_CRYPT_01501] [SWS_CRYPT_01502] [SWS_CRYPT_01503] [SWS_CRYPT_01504] [SWS_CRYPT_01505] [SWS_CRYPT_01506] [SWS_CRYPT_01507] [SWS_CRYPT_01520] [SWS_CRYPT_01651] [SWS_CRYPT_01652] [SWS_CRYPT_01653] [SWS_CRYPT_01654] [SWS_CRYPT_01655] [SWS_CRYPT_01656] [SWS_CRYPT_01658] [SWS_CRYPT_01659] [SWS_CRYPT_01662] [SWS_CRYPT_01663] [SWS_CRYPT_01664] [SWS_CRYPT_01665] [SWS_CRYPT_01666] [SWS_CRYPT_01667] [SWS_CRYPT_10042] [SWS_CRYPT_20742] [SWS_CRYPT_20744] [SWS_CRYPT_23600] [SWS_CRYPT_23601] [SWS_CRYPT_23700] [SWS_CRYPT_23701] [SWS_CRYPT_23716] [SWS_CRYPT_23717] [SWS_CRYPT_23801] [SWS_CRYPT_23802] [SWS_CRYPT_24001] [SWS_CRYPT_24011] [SWS_CRYPT_24012] [SWS_CRYPT_24013] [SWS_CRYPT_24014]

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_02706] [SWS_CRYPT_02726] [SWS_CRYPT_10042] [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_20813] [SWS_CRYPT_21000] [SWS_CRYPT_21001] [SWS_CRYPT_21011] [SWS_CRYPT_21012] [SWS_CRYPT_21013] [SWS_CRYPT_22200] [SWS_CRYPT_22700] [SWS_CRYPT_22701] [SWS_CRYPT_22702] [SWS_CRYPT_22711] [SWS_CRYPT_22712] [SWS_CRYPT_22713] [SWS_CRYPT_23200] [SWS_CRYPT_23201] [SWS_CRYPT_23215] [SWS_CRYPT_23216]

Requirement	Description	Satisfied by
[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_01205] [SWS_CRYPT_01207] [SWS_CRYPT_01208] [SWS_CRYPT_01209] [SWS_CRYPT_01210] [SWS_CRYPT_01211] [SWS_CRYPT_01213] [SWS_CRYPT_01218] [SWS_CRYPT_01219] [SWS_CRYPT_01220] [SWS_CRYPT_10042] [SWS_CRYPT_20319] [SWS_CRYPT_20746] [SWS_CRYPT_22100] [SWS_CRYPT_22101] [SWS_CRYPT_22115] [SWS_CRYPT_22116] [SWS_CRYPT_22117] [SWS_CRYPT_22119] [SWS_CRYPT_23300] [SWS_CRYPT_23301] [SWS_CRYPT_23302] [SWS_CRYPT_23311]
[RS_CRYPTO_-02204]	The Crypto Stack shall provide interfaces to use digital signature primitives	[SWS_CRYPT_00902] [SWS_CRYPT_02400] [SWS_CRYPT_02403] [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_02421] [SWS_CRYPT_02422] [SWS_CRYPT_10042] [SWS_CRYPT_20003] [SWS_CRYPT_20319] [SWS_CRYPT_20754] [SWS_CRYPT_20755] [SWS_CRYPT_20756]

Requirement	Description	Satisfied by
		[SWS_CRYPT_20757] [SWS_CRYPT_22119] [SWS_CRYPT_22200] [SWS_CRYPT_22201] [SWS_CRYPT_22215] [SWS_CRYPT_22216] [SWS_CRYPT_23200] [SWS_CRYPT_23201] [SWS_CRYPT_23300] [SWS_CRYPT_23301] [SWS_CRYPT_23302] [SWS_CRYPT_23311] [SWS_CRYPT_23500] [SWS_CRYPT_23501] [SWS_CRYPT_23511] [SWS_CRYPT_23512] [SWS_CRYPT_23513] [SWS_CRYPT_23514] [SWS_CRYPT_24100] [SWS_CRYPT_24101] [SWS_CRYPT_24111] [SWS_CRYPT_24112] [SWS_CRYPT_24113] [SWS_CRYPT_24114]
[RS_CRYPTO_-02205]	The Crypto Stack shall provide interfaces to use hashing primitives	[SWS_CRYPT_00901] [SWS_CRYPT_00903] [SWS_CRYPT_00905] [SWS_CRYPT_00906] [SWS_CRYPT_00907] [SWS_CRYPT_00908] [SWS_CRYPT_00909] [SWS_CRYPT_00910] [SWS_CRYPT_00919] [SWS_CRYPT_10042] [SWS_CRYPT_20747] [SWS_CRYPT_21100] [SWS_CRYPT_21101] [SWS_CRYPT_21115] [SWS_CRYPT_21116] [SWS_CRYPT_21117] [SWS_CRYPT_23300] [SWS_CRYPT_23301] [SWS_CRYPT_23302] [SWS_CRYPT_23311]

Requirement	Description	Satisfied by
[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_10042] [SWS_CRYPT_20741] [SWS_CRYPT_22900] [SWS_CRYPT_22901] [SWS_CRYPT_22911] [SWS_CRYPT_22912] [SWS_CRYPT_22914] [SWS_CRYPT_22915] [SWS_CRYPT_30098]
[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_10042] [SWS_CRYPT_20100] [SWS_CRYPT_20101] [SWS_CRYPT_20316] [SWS_CRYPT_20745]
[RS_CRYPTO_-02208]	The Crypto Stack shall provide interfaces to use symmetric key wrapping primitives	[SWS_CRYPT_02100] [SWS_CRYPT_02101] [SWS_CRYPT_02102] [SWS_CRYPT_02103] [SWS_CRYPT_02104] [SWS_CRYPT_02105] [SWS_CRYPT_02106] [SWS_CRYPT_02107] [SWS_CRYPT_02120] [SWS_CRYPT_10042] [SWS_CRYPT_20743] [SWS_CRYPT_24000]

Requirement	Description	Satisfied by
[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_10042] [SWS_CRYPT_20752] [SWS_CRYPT_20753] [SWS_CRYPT_21400] [SWS_CRYPT_21800] [SWS_CRYPT_21801]
[RS_CRYPTO_-02302]	The Crypto Stack API shall support a streaming approach	[SWS_CRYPT_01657] [SWS_CRYPT_01661] [SWS_CRYPT_10701] [SWS_CRYPT_10710] [SWS_CRYPT_10750] [SWS_CRYPT_10751] [SWS_CRYPT_10752] [SWS_CRYPT_10753] [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_23617] [SWS_CRYPT_23618] [SWS_CRYPT_23619] [SWS_CRYPT_23620] [SWS_CRYPT_23621] [SWS_CRYPT_23622] [SWS_CRYPT_23625] [SWS_CRYPT_23626] [SWS_CRYPT_23634] [SWS_CRYPT_23635] [SWS_CRYPT_23715] [SWS_CRYPT_24714] [SWS_CRYPT_24715]

Requirement	Description	Satisfied by
[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_01660] [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]
[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]

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

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

Requirement	Description	Satisfied by
		[SWS_CRYPT_40630] [SWS_CRYPT_40631] [SWS_CRYPT_40632] [SWS_CRYPT_40633] [SWS_CRYPT_40634] [SWS_CRYPT_40635] [SWS_CRYPT_40636] [SWS_CRYPT_40640] [SWS_CRYPT_40700] [SWS_CRYPT_40701] [SWS_CRYPT_40702] [SWS_CRYPT_40711] [SWS_CRYPT_40800] [SWS_CRYPT_40801] [SWS_CRYPT_40802] [SWS_CRYPT_40811] [SWS_CRYPT_40900]
[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_03900] [SWS_CRYPT_03902] [SWS_CRYPT_03904] [SWS_CRYPT_03905] [SWS_CRYPT_03906] [SWS_CRYPT_03910] [SWS_CRYPT_20651] [SWS_CRYPT_20711] [SWS_CRYPT_20712]
[RS_CRYPTO_-02309]	The Crypto Stack API shall support the run-time configurable usage style	[SWS_CRYPT_20103] [SWS_CRYPT_20412] [SWS_CRYPT_20516] [SWS_CRYPT_20652] [SWS_CRYPT_21415] [SWS_CRYPT_21416] [SWS_CRYPT_21514] [SWS_CRYPT_21715] [SWS_CRYPT_21817] [SWS_CRYPT_21818] [SWS_CRYPT_22213] [SWS_CRYPT_22214] [SWS_CRYPT_23213] [SWS_CRYPT_23214] [SWS_CRYPT_23312] [SWS_CRYPT_23611] [SWS_CRYPT_23612] [SWS_CRYPT_23624] [SWS_CRYPT_23711] [SWS_CRYPT_23712] [SWS_CRYPT_23713] [SWS_CRYPT_24411] [SWS_CRYPT_24412] [SWS_CRYPT_24413]

Requirement	Description	Satisfied by
		[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]
[RS_CRYPTO_-02310]	The Crypto Stack API shall support an efficient mechanism of error states notification	[SWS_CRYPT_00104] [SWS_CRYPT_10099] [SWS_CRYPT_19902] [SWS_CRYPT_19903] [SWS_CRYPT_19950] [SWS_CRYPT_19951] [SWS_CRYPT_19953] [SWS_CRYPT_19954]

Requirement	Description	Satisfied by
[RS_CRYPTO_-02401]	The Crypto Stack should support a joint usage of multiple back-end cryptography providers including ones with non-extractable keys	[SWS_CRYPT_00005] [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_30131] [SWS_CRYPT_30403] [SWS_CRYPT_40911]
[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_10005] [SWS_CRYPT_30400] [SWS_CRYPT_30401] [SWS_CRYPT_30402]
[SWS_CORE_-10910]	ErrorDomain exception base type	[SWS_CRYPT_19905] [SWS_CRYPT_19906]
[SWS_CORE_-10934]	ErrorDomain subclass Exception symbol	[SWS_CRYPT_19904]

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:

- CryptoProvider
- KeyStorageProvider
- X.509 Certificate Management Provider

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

7.1 Functional Cluster Lifecycle

7.1.1 Startup

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

[SWS_CRYPT_00101]{DRAFT} ⌈When ara::core::Intitialize is called, the FC Crypto shall read in the manifest information and prepare the access structures to CryptoProvider and CryptoKeySlot that are defined in the manifest.⌋*(RS_CRYPTO_02110)*
Hint: Access structures may encompass the communication channel between the application process and the stack process or other resource required by the CryptoAPI.

7.1.2 Shutdown

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

[SWS_CRYPT_00103]{DRAFT} ⌈When ara::core::Deinitilize is called, the FC Crypto shall ensure that all associated persist operations in this context of this application are executed successfully and no new persist operations are started.⌋*(RS_CRYPTO_02004)*

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

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

7.2 Architectural concepts

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

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

- Authentication: FC Crypto provides mechanisms that allow adaptive applications or functional clusters to prove their identity to other applications or functional clusters.
- Non-Repudiation: FC Crypto supports the concept of non-repudiation, where someone cannot deny the validity of something.
- Confidentiality: FC Crypto allows to keep information private. Cryptographic systems were originally developed to function in this capacity. Whether it be system or user specific data sent during system debugging or tracing, or storing confidential vehicle / ECU data, encryption can assure that only users who have access to the appropriate key will get read access to the data plaintext.

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

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

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

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

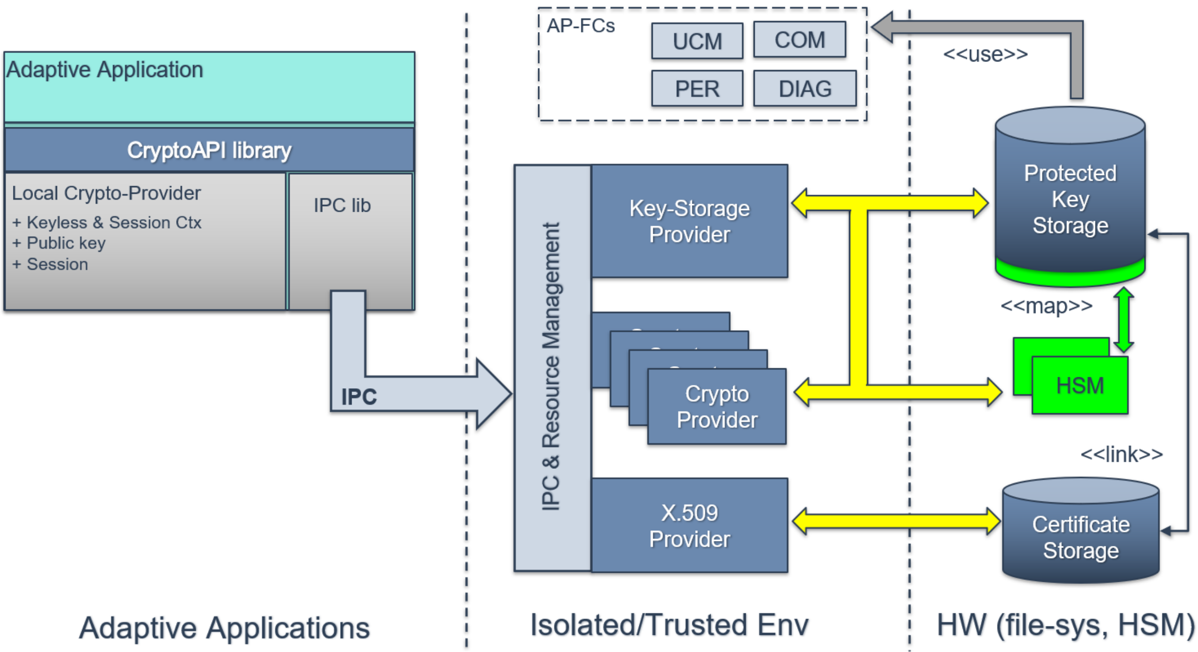

Figure 7.1: High-level CryptoAPI architecture

7.2.1 Integration with Identity and Access Management

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

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

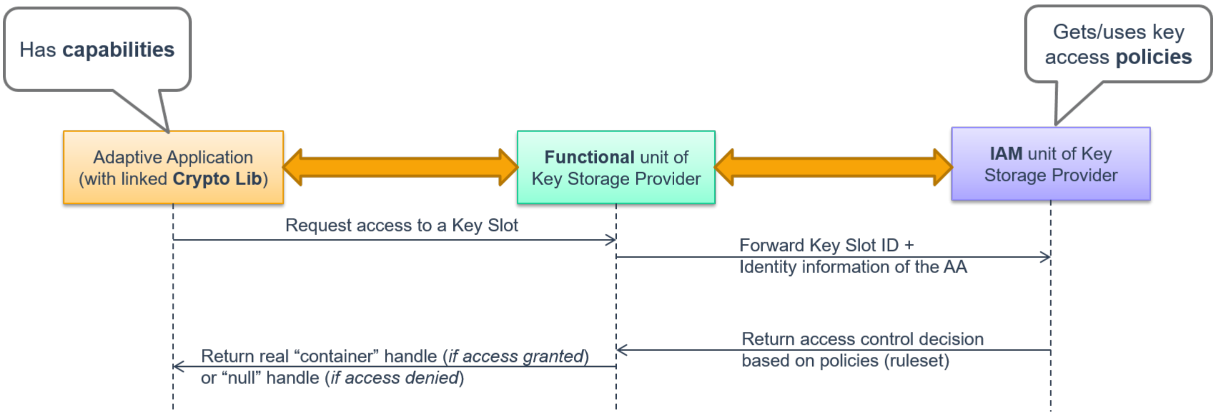

Figure 7.2: Interaction with IAM

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

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

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

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

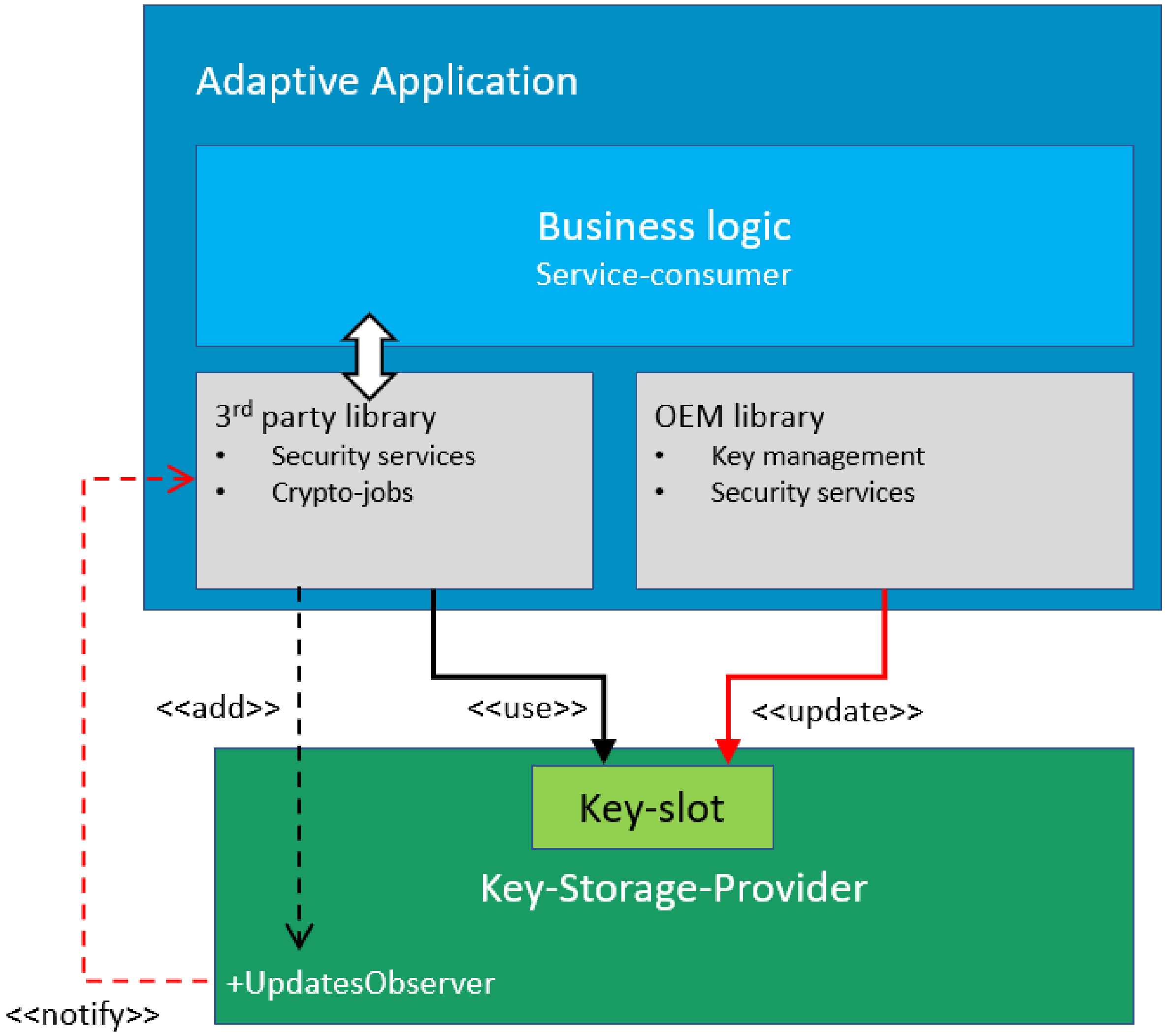

Figure 7.3: Concurrent access to a single Key Slot

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

7.2.2 Integration into AUTOSAR

The overall architecture is described in 7.2. The FC Crypto provides its service to all AUTOSAR elements, such as untrusted Adaptive Applications or trusted system services (functional clusters). From cryptographic service point of view both could be treated equally. The integraton of FC Crypto into AUTOSAR is described in Figure 7.4.

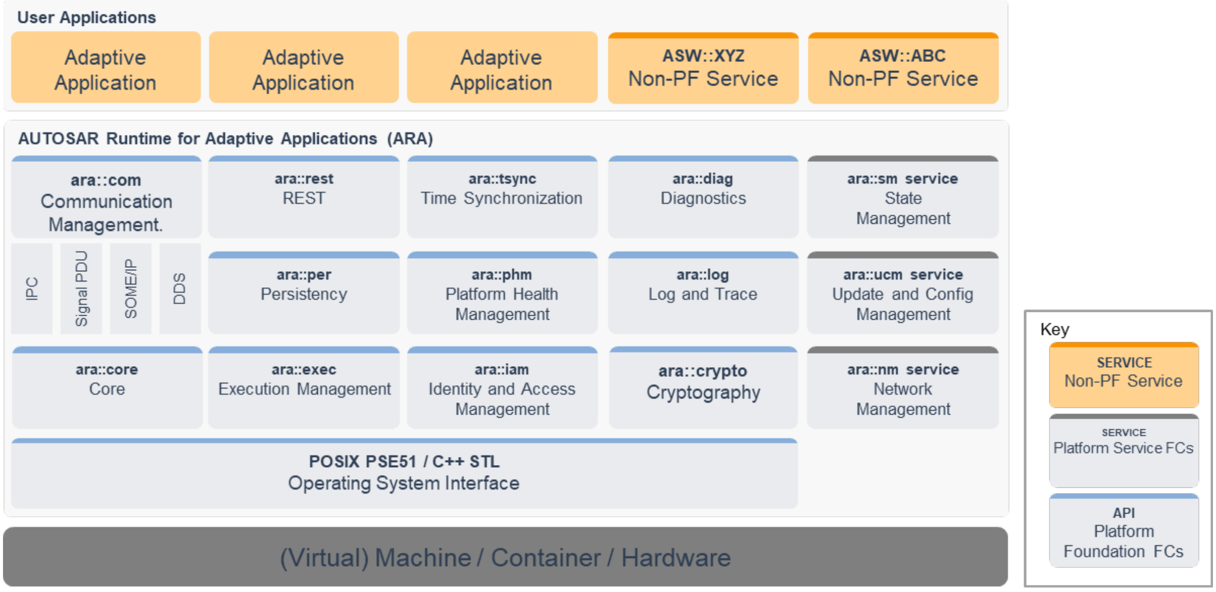

Figure 7.4: Integration into AUTOSAR

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

- **application**: the application has exclusive access to this key slot. It is able to import/export, update/delete and use the contained key-material. No other application or functional cluster may access this Key Slot.
- **machine**: this type of Key Slot is defined by the adaptive machine and may be used by the functional cluster for which it is configured. Additionally, the Key Slot may be assigned to a single Adaptive Application that is then able to manage the contained key-material.

Figure 7.5 gives an example for the use of machine and application key slots.

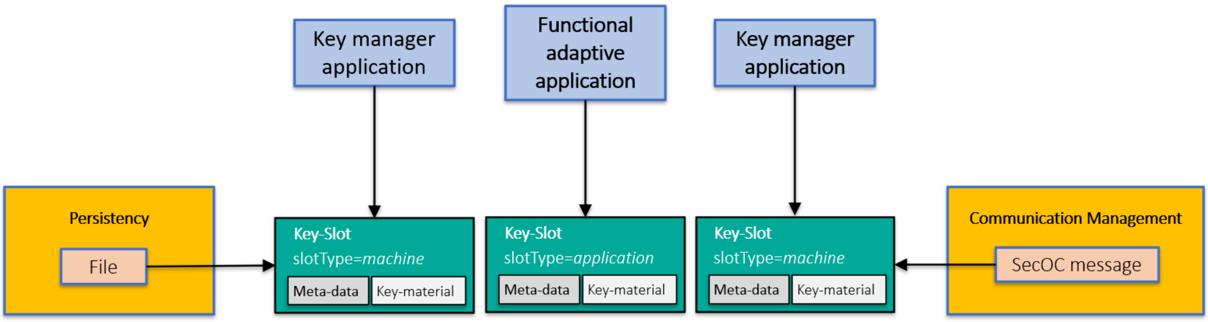

Figure 7.5: Key Slot types and usages

7.2.3 Application level

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

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

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

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

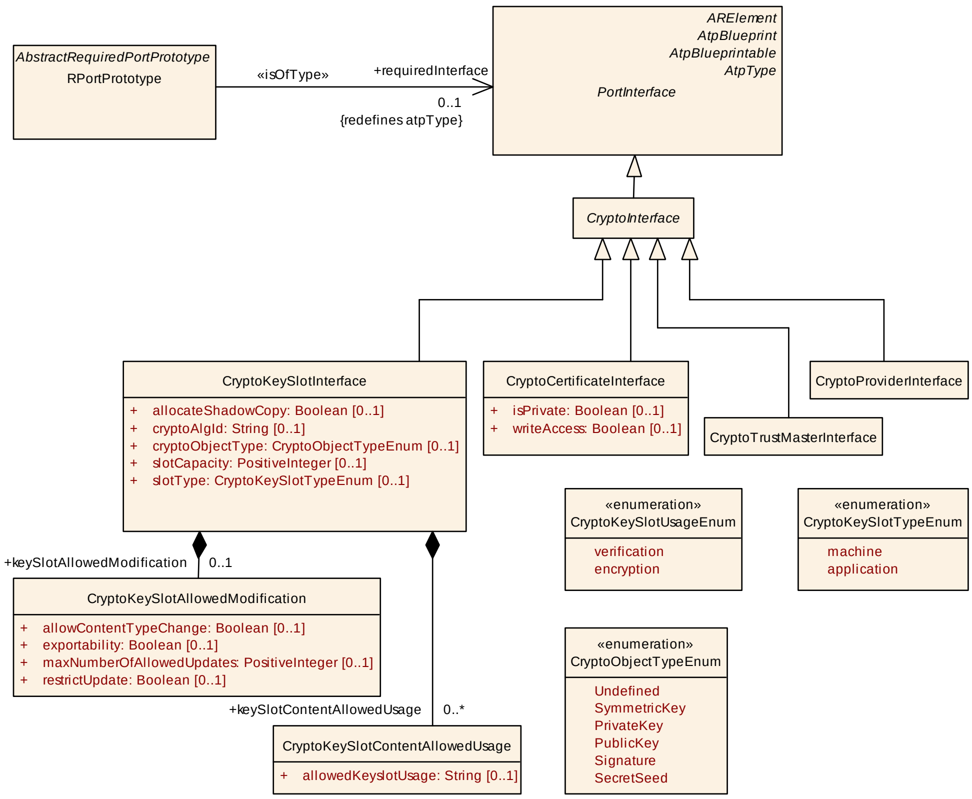

Figure 7.6: Application interface

7.2.4 System service level

Some `Adaptive Platform Services` such as update and configuration, communication, persistency or diagnostics also require cryptographic services as part of their functionality. If key-material is needed and must be configurable by an `Adaptive Application` (e.g. OEM key manager), the platform shall specify a `Key Slot` of slot-Type *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. **CryptoProvider** (CP, namespace `ara::crypto::cryp`) is responsible for implementation of all supported cryptographic primitives. FC Crypto may support multiple instances of the `CryptoProvider`s. Each instance of `CryptoProvider` represents single holistic software- or hardware-based implementation of some set of cryptographic algorithms. Each `CryptoProvider` 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 `CryptoProvider`) 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 `CryptoProvider` and KSP implementation details, therefore CMP and `CryptoProvider`/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 `CryptoProvider` 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. When the systems provide multiple hardware elements and or software elements, then the same amount of Crypto Providers exists as hardware and software elements are in the system.

[SWS_CRYPT_00004]{DRAFT} ⌈Based on this definition, each Crypto Provider shall implement its supporting cryptographic primitives and is represented by an instance of CryptoProvider. The instance of the CryptoProvider may not export all supporting cryptographic primitives to external users. This can be useful when a cryptographic library contains old, outdated, or weak cryptographic primitives, that not all primitives or functionality is represented by the instance. However, this implementation detail shall be documented and communicated to the users.⌋*(RS_CRYPTO_02305)*

The application designer is able to define the request to use a CryptoProvider with the creation of an RPortPrototype that is typed by a CryptoProviderInterface. The integrator will map this RPortPrototype to a concrete CryptoProvider representation in the manifest with the CryptoProviderToPortPrototypeMapping. This mapping takes also the Process into account since the Executable that contains the SwComponent that in turn contains the RPortPrototype may be started several times.

[SWS_CRYPT_00005]{DRAFT} ⌈The FC Crypto may support multiple instances of the Crypto Providers. Applications and services can access these instances by CryptoProviderInterface.⌋*(RS_CRYPTO_02401)*

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

[SWS_CRYPT_00007]{DRAFT} ⌈FC Crypto shall isolate all key material, which are used during processing, from unauthorized access.⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002)*

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

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

[SWS_CRYPT_00500]{DRAFT} ⌈Each CryptoProvider may provide zero, one, or more Random Number Generator (RNG) implementations. Therefore, the FC Crypto provides RandomGeneratorCtx context. The CryptoAPI provides the CreateRandomGeneratorCtx to generate this context.⌋*(RS_CRYPTO_02206)*

[SWS_CRYPT_00501]{DRAFT} ⌈If a CryptoProvider provides one or more random generator implementations, one random generator implementation shall be documented as the default and a corresponding RandomGeneratorCtx shall be returned when CreateRandomGeneratorCtx() is called with algId == kAlgIdDefault.

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

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

Each RandomGeneratorCtx may either rely on state local to the RandomGeneratorCtx instance only, or may rely on global state shared among different 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 RandomGeneratorCtx.

[SWS_CRYPT_00502]{DRAFT} ⌈If a RandomGeneratorCtx uses global state, calls to its methods Seed(), SetKey(), and AddEntropy() shall return false without modifying the global state.⌋*(RS_CRYPTO_02206)*

[SWS_CRYPT_00503]{DRAFT} ⌈Seed() and SetKey() shall return false without modifying the global state, if they are called with a SymmetricKey or a SecretSeed without the allowed usage flag kAllowRngInit.⌋*(RS_CRYPTO_02206)*

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

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

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

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

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

[SWS_CRYPT_00506]{DRAFT} ⌈If `CreateRandomGeneratorCtx()` is called to create a global-state `RandomGeneratorCtx`, the parameter `initialize` shall be ignored by `FC Crypto` and the requested `RandomGeneratorCtx` shall be returned without modification of its state.⌋*(RS_CRYPTO_02206)*

A `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, `Generate()` shall return errors.

[SWS_CRYPT_00507]{DRAFT} ⌈If a call to `Generate()` of a global-state `RandomGeneratorCtx` cannot be served with the requested number of random bytes, `SecurityErrorDomain::kBusyResource` shall be returned.⌋*(RS_CRYPTO_02206)*

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

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

7.4.1.2 Key Derivation Function (KDF)

According to [8], [9], [10], and [11] 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_00601]{DRAFT} ⌈Calling the function CreateKeyDerivationFunctionCtx of a Crypto Provider shall return an instance of KeyDerivationFunctionCtx identified by the parameter AlgId.⌋*(RS_CRYPTO_02101)*

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

[SWS_CRYPT_00602]{DRAFT} **Hash based KDF** ⌈During the initialization phase of the context, FC Crypto shall allow to parametrize a hash function as the used key derivation function. This is done by the algorithm identifier.⌋*(RS_CRYPTO_02101)*

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

[SWS_CRYPT_00608]{DRAFT} ⌈The FC Crypto shall support salting to improve the secrecy of the derived key.⌋*(RS_CRYPTO_02101)*

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

[SWS_CRYPT_00609]{DRAFT} **Good entropy smoothing function** ⌈The FC Crypto shall allow to derive slave key material (secret seed) from provided master key material with optional public or secret salt. To use secret salt, an application or functional cluster uses the AddSecretSalt to provide a secrete salt value to the context. The CryptoAPI also supports adding a public salt by AddSalt. Deriving a slave key is done by the given target symmetric algorithm identifier, which also defines the target seed-length.⌋*(RS_CRYPTO_02101)*

[SWS_CRYPT_00610]{DRAFT} ⌈The FC Crypto shall allow to specify the number of iterations for generating keys. When no number or zero is given, the default number of iterations is used. Otherwise, the provided iterations is used. However, the implementation can restrict minimal and / or maximal value of the iterations number. The CryptoAPI this functionality with ConfigIterations.⌋*(RS_CRYPTO_02101)*

[SWS_CRYPT_00611]{DRAFT} ⌈The FC Crypto shall allow to set the properties of the derived key as follow:

- ”session” (or ”temporary”) attribute defines the lifetime for the derived key.
- ”exportable” defines if a derived key can be stored outside the system

The CryptoAPI provides the interface DeriveKey, where the application or functional cluster can choose between a session or exportable lifetime of the derived key.⌋*(RS_CRYPTO_02101)*

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

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

⌋(*RS_CRYPTO_02101*)

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 cluster 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_00901]{DRAFT} ⌈Calling the function `CreateHashFunctionCtx` of a `Crypto Provider` shall return an instance of `HashFunctionCtx` identified by the parameter `AlgId`.⌋(*RS_CRYPTO_02205*) The `AlgId` identifier represents the common name as defined in chapter 7.5.

[SWS_CRYPT_00902]{DRAFT} ⌈The `HashFunctionCtx` shall implement hashing.⌋(*RS_CRYPTO_02204*)

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

[SWS_CRYPT_00908]{DRAFT} **Start** ⌈The function `Start` shall clear the current hash value and initialize the context with the provided `IV`.

- `Start` shall return a `SecurityErrorDomain::kInvalidInputSize` error, if the size of the provided `IV` is not supported by the configured context `AlgId`.
- `Start` shall return a `SecurityErrorDomain::kUnsupported` error, if the configured context `AlgId` does not support an `IV`.
- `Start` shall return a `SecurityErrorDomain::kMissingArgument` error, if the configured context `AlgId` expected an `IV` but none was provided.

⌋(*RS_CRYPTO_02205*)

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

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

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

- Commonly in practice the entire hash object is not in one contiguous segment available. Instead, often parts are used independently as given by the `HMAC` function for example. Here, the inner hash is some preprocessed keying material, followed by the message being `MAC`'ed. Therefore, a temporary buffer consisting of the `HMAC` inner key ("ipad") and the message can be created. However, this is an overhead.
- The incrementally creation allows to run the hash implementation in memory complexity O(1). The needed memory space for calculation is independent of input size. This is very easy to do with current hash function, such as `SHA-2` and `SHA-3`, where, with a small amount of side memory, the hashing processes the message in pieces.

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

[SWS_CRYPT_00905]{DRAFT} **Update** ⌈The function `Update` shall implement the configured hash algorithm calculation.⌋(*RS_CRYPTO_02205*)

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

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

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

- Finish shall return a SecurityErrorDomain::kProcessingNotStarted error, if Start has not been successfully called before.
- Finish shall return a SecurityErrorDomain::kInvalidUsageOrder error, if Update has not been called successfully after the last call to Start.

⌋*(RS_CRYPTO_02205)*

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

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

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

[SWS_CRYPT_00919]{DRAFT} **Signalization of missing finalization error** ⌈GetDigest shall return a SecurityErrorDomain::kProcessingNotStarted error, if Finish has not been called for the current hash value calculation.⌋*(RS_CRYPTO_02205)*

7.4.1.4 Message Authentication Code (MAC)

According to the ISO-9797 [12] 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 [13] and in [14]. 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

[SWS_CRYPT_01200]{DRAFT} ⌈The FC Crypto shall support Message Authentication Code generation as described in [13] and in [14]. Therefore, the FC Crypto provides the MessageAuthnCodeCtx context. The CryptoAPI provides the CreateMessageAuthnCodeCtx to generate this context. This context needs an identifier to specify the used cryptographic algorithm. The context can deliberately combine two or more cryptographic primitives.⌋*(RS_CRYPTO_02203)*

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.

[SWS_CRYPT_01201]{DRAFT} **Startup** ⌈The context handles two different use cases, when an application or functional cluster Start the 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.

⌋*(RS_CRYPTO_02203)*

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

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

[SWS_CRYPT_01203]{DRAFT} ⌈It shall be possible to use a SecretSeed as an IV during initialization.⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_01204]{DRAFT} **Update** ⌈The function Update shall implement the configured hash algorithm calculation.⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_01205]{DRAFT} **Update** ⌈The user application shall be able to call Update multiple times, each time providing a new chunk of data. Update shall update the hash value calculation with each new chunk. `Update` shall return a `SecurityErrorDomain::kProcessingNotStarted` error, if `Start` has not been called before.⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_01207]{DRAFT} **Finish** ⌈The `Finish` function shall finalize the hash value calculation and return the hash value, i.e. no more data may be provided by `Update`.

- `Finish` shall return a `SecurityErrorDomain::kProcessingNotStarted` error, if `Start` has not been successfully called before.
- `Finish` shall return a `SecurityErrorDomain::kInvalidUsageOrder` error, if `Update` has not been called successfully after the last call to `Start`.

⌋*(RS_CRYPTO_02203)*

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

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

[SWS_CRYPT_01210]{DRAFT} ⌈The context allows to request the calculated digest as part or as whole. With the `CryptoAPI` call `GetDigest` the calling application or functional cluster will get the hashed output. The `CryptoAPI` allows also to specific an offset. This offset informs the `Crypto Provider` where the position of first byte of digest is that should be placed to the output buffer.⌋*(RS_CRYPTO_02203)*

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

[SWS_CRYPT_01211]{DRAFT} ⌈The `CryptoAPI` provides `SetKey`, which enables the context to set (deploy) a key.⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_01213]{DRAFT} **Verify** ⌈The `CryptoAPI` shall `Check` if previous calculated and internally stored `MAC` is valid to an expected "signature" object. Validation is successful, if value and meta-information of the provided "signature" object is identical to calculated digest and current configuration of the context.⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_01218]{DRAFT} **Missing required `IV` error** ⌈If an application or functional cluster wants to use a cryptographic primitive, which needs an `IV`, and does not provide one, the `Crypto Provider` will throw a distinct error.⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_01219]{DRAFT} **Signalization of missing finalization error** ⌈If an application or functional cluster wants to read the calculated MAC or compare it with another MAC before the Finalize is called, the Crypto Provider will throw a distinct error.⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_01220]{DRAFT} **Signalization of error** ⌈GetDigest shall return a SecurityErrorDomain::kProcessingNotStarted error, if Finish has not been called for the current digest value calculation.⌋*(RS_CRYPTO_02203)*

7.4.1.5 Symmetric encryption

Symmetric encryption uses a shared secret (e.g., share key) to encrypt and / or decrypt an information. Without knowing the key, the information cannot be understood by anyone. Two groups categorize symmetric cryptographic algorithms:

1. Block Cipher: An information with a fixed length is encrypted. The system holds the data in its memory as its waits for complete blocks.

2. Stream Cipher: The 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 SymmetricKey to encrypt an input data. Block Ciphers are commonly used to protect data at rest, such as on file systems.

[SWS_CRYPT_01501]{DRAFT} ⌈The FC Crypto shall support Block Cipher cryptography with the BlockCipherCtx. The CryptoAPI provides the CreateBlockCipherCtx to generate this context. This context needs an identifier to specify the used cryptographic algorithm.⌋*(RS_CRYPTO_02107, RS_CRYPTO_02201)*

This identifier is encoded with the common name as defined in chapter 7.5.

[SWS_CRYPT_01502]{DRAFT} ⌈The CryptoAPI provides an interface, which enables the context to set (deploy) a key.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01503]{DRAFT} ⌈Additionally, the CryptoAPI shall allow to set a "direction" indicator. This indicator defines the transformation usage, such as encryption, decryption, signature calculation, or signature verification.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01504]{DRAFT} ⌈The CryptoAPI allows to query the current configuration of the transformation.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01505]{DRAFT} ⌈The CryptoAPI shall provide an encryption of a plaintext or decryption of a ciphertext according to the configuration and used block cipher algorithm.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01506]{DRAFT} ⌈Some Block Cipher need an input data length be 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, [15, PKCS5],[16, PKCS5], [17, PKCS7], or [18, ANSI X9.23].⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01507]{DRAFT} ⌈After decrypting the padding shall be removed.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01520]{DRAFT} **Signalization of error** ⌈By conventions, if any algorithm fails the FC Crypto shall provide a distinct error.⌋*(RS_CRYPTO_02201)*

7.4.1.5.2 Stream Cipher

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

[SWS_CRYPT_01651]{DRAFT} ⌈The FC Crypto shall support Stream Cipher cryptography with the StreamCipherCtx. The CryptoAPI provides the CreateStreamCipherCtx to generate this context. This context needs an identifier to specify the used cryptographic algorithm.⌋*(RS_CRYPTO_02201)* This identifier is encoded with the common name as defined in chapter 7.5.

[SWS_CRYPT_01652]{DRAFT} ⌈The CryptoAPI provides an interface, which enables the context to set (deploy) a key.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01653]{DRAFT} ⌈Additionally, the CryptoAPI shall allow to set a "direction" indicator. This indicator defines the transformation usage, such as encryption, decryption, signature calculation, or signature verification.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01654]{DRAFT} ⌈The FC Crypto shall support to process and generate data streams for Stream Ciphers. The CryptoAPI allows to Start the data stream depending on the used cryptographic algorithm.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01655]{DRAFT} ⌈In order to reuse keys an Initialization Vector (IV) or Nonce value shall be used. If IV size is greater than the maximally by the algorithm supported length, then the FC Crypto may use the leading bytes only.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01656]{DRAFT} ⌈The FC Crypto allows to process initial parts of message aligned to the block-size boundary.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01657]{DRAFT} ⌈The CryptoAPI allows also to Update non-final part of message, that is not aligned to the block-size boundary.⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_01658]{DRAFT} ⌈The CryptoAPI shall finalize the encryption or decryption of data stream by calling Finish.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01659]{DRAFT} ⌈It shall be possible to process a whole message in one function call.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01660]{DRAFT} ⌈The CryptoAPI shall provide a seek functionality. This allows to set the position of the next byte within the stream of the encryption / decryption gamma. An offset value in bytes is used setting the relative, to starting point, or current position in the gamma stream. It is possible to indicate if the starting point for positioning is within the stream from the starting point or from the current position.⌋*(RS_CRYPTO_02304)*

Some operation modes of specific Stream Ciphers are seekable, e.g., [19, CTR], [20, Salsa20], or [21, Trivium], and others are not. Seekable means that the user can efficiently seek to any position in the key 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_01661]{DRAFT} ⌈The CryptoAPI shall provide the IsSeekableMode function such that the user can query the information if a Stream Cipher is seekable or not.⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_01662]{DRAFT} ⌈The CryptoAPI allows to query the current mode of operation, which can be either bytewise or block-by-block. In bytewise mode the messages are processed byte-by-byte (without padding up to the block boundary). Otherwise, messages will be processed block-by-block (only full blocks can be processed, the padding is mandatory).⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01663]{DRAFT} ⌈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, [15, PKCS5], [16, PKCS5], [17, PKCS7], or [18, ANSI X9.23].⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01664]{DRAFT} ⌈After decrypting the padding shall be removed.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01665]{DRAFT} ⌈The CryptoAPI allows to query the current configuration of the transformation. Direct indicates for example encryption or signature calculation. Reverse, on the other hand, indicates decryption or signature verification.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01666]{DRAFT} **Signalization of error** ⌈By conventions, if any algorithm fails the FC Crypto shall provide a distinct error.⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_01667]{DRAFT} **Failing streams** ⌈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.⌋*(RS_CRYPTO_02201)*

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.

[SWS_CRYPT_01800]{DRAFT} ⌈UpdateAssociatedData shall return a SecurityErrorDomain::kInvalidUsageOrder error, if ProcessConfidentialData has already been called.⌋*(RS_CRYPTO_02207)*

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

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

[SWS_CRYPT_01803]{DRAFT} ⌈ProcessConfidentialData shall update the calculation of the MAC with the confidential data.⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_01804]{DRAFT} ⌈If the transformation direction is CryptoTransform::kEncrypt, ProcessConfidentialData shall also encrypt the provided plaintext data and return the ciphertext.⌋*(RS_CRYPTO_02207)*

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

[SWS_CRYPT_01806]{DRAFT} ⌈Calling the function CreateAuthCipherCtx of a Crypto Provider shall return an instance of AuthCipherCtx identified by the parameter AlgId.⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_01807]{DRAFT} ⌈The SetKey interface of the AuthCipherCtx shall check the allowed-usage flags of the key parameter provided.

- SetKey shall return a SecurityErrorDomain::kUsageViolation error, if kAllowDataEncryption is not set and the transformation direction is CryptoTransform::kEncrypt.
- SetKey shall return a SecurityErrorDomain::kUsageViolation error, if kAllowDataDecryption is not set and the transformation direction is CryptoTransform::kDecrypt.

⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_01808]{DRAFT} ⌈The function `Start` shall initialize the transformation using the provided `IV` or `Nonce`.

- `Start` shall return a `SecurityErrorDomain::kUninitializedContext` error, if `SetKey` has not been called before.
- `Start` shall return a `SecurityErrorDomain::kInvalidInputSize` error, if the provided data is insufficient.
- `Start` shall return a `SecurityErrorDomain::kUnsupported` error, if the `AlgId` specified does not support an `IV` or a `Nonce`.
- `Start` shall return a `SecurityErrorDomain::kUsageViolation` error, if a `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 `SetKey` function (`CryptoTransform::kEncrypt` or `CryptoTransform::kDecrypt`).

⌋*(RS_CRYPTO_02207)*

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

7.4.1.7 Key Wrapping

Key Wrapping (as defined in [22] and [23]) 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.

[SWS_CRYPT_02100]{DRAFT} ⌈The `FC Crypto` shall support key wrapping. Therefore, the `SymmetricKeyWrapperCtx` is provided. The `CryptoAPI` provides the `CreateSymmetricKeyWrapperCtx` interface to create this context. The identifier of a target algorithm is needed to create the context.⌋*(RS_CRYPTO_02208)*

This identifier is provided as common name as described in chapter 7.5.

[SWS_CRYPT_02101]{DRAFT} ⌈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.⌋*(RS_CRYPTO_02208)*

[SWS_CRYPT_02102]{DRAFT} **Wrapping** ⌈The context shall execute the `Encapsulate` operation for the provided key material in `plaintext`. The `KEK` itself is not needed, because it is already defined by the context.⌋*(RS_CRYPTO_02208)*

[SWS_CRYPT_02103]{DRAFT} **Unwrapping** ⌈The context shall support the reverse operation. In order to execute the `DecapsulateKey` operation the context needs the ciphertext, i.e. wrapped key. The KEK itself is not needed, because it is already defined by the context.⌋*(RS_CRYPTO_02208)*

[SWS_CRYPT_02104]{DRAFT} ⌈The FC Crypto shall allow to change the cryptographic algorithm, which was given during context creation, for unwrapping a ciphertext.⌋*(RS_CRYPTO_02208)*

[SWS_CRYPT_02105]{DRAFT} ⌈Via a CryptoAPI call the FC Crypto shall embed the new unwrapped key into its internal key management infrastructure. Additionally the transformation types of the new unwrapped key can be set by a CryptoAPI interface. This transformation type is also stored in the FC Crypto as property of the new unwrapped key.⌋*(RS_CRYPTO_02208)*

[SWS_CRYPT_02106]{DRAFT} ⌈The FC Crypto allows to specify additional usage parameter for the key or seed. Depending on the used context, the CryptoAPI method sets the specified usage parameter in the key or seed attribute. The usage parameters indicates the transformation types in which the target key or seed can be used.⌋*(RS_CRYPTO_02208)*

[SWS_CRYPT_02107]{DRAFT} ⌈Depending on the usage of padding and the nature of padding, the expected granularity of the target key (block size) shall be provided.⌋*(RS_CRYPTO_02208)*

[SWS_CRYPT_02120]{DRAFT} **Signalization of error** ⌈By conventions, if any algorithm fails the FC Crypto shall provide a distinct error.⌋*(RS_CRYPTO_02208)*

7.4.1.8 Digital signatures

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

[SWS_CRYPT_02400]{DRAFT} ⌈Calling the function CreateVerifierPublicCtx of a Crypto Provider shall return an instance of VerifierPublicCtx identified by the parameter AlgId.⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02408]{DRAFT} ⌈Calling the function CreateSignerPrivateCtx of a Crypto Provider shall return an instance of SignerPrivateCtx identified by the parameter AlgId.⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02409]{DRAFT} ⌈Calling the function CreateSigEncodePrivateCtx of a Crypto Provider shall return an instance of SigEncodePrivateCtx identified by the parameter AlgId.⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02410]{DRAFT} ⌈Calling the function CreateMsgRecoveryPublicCtx of a Crypto Provider shall return an instance of MsgRecoveryPublicCtx identified by the parameter AlgId.⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02411]{DRAFT} ⌈The MsgRecoveryPublicCtx shall implement digital signature verification with message recovery according to [24].⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02412]{DRAFT} ⌈The SigEncodePrivateCtx shall implement digital signature generation with message encoding according to [24].⌋*(RS_CRYPTO_02204)*

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

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

[SWS_CRYPT_01820]{DRAFT} ⌈The SetKey interface of the SignerPrivateCtx shall check the allowed-usage flags of the key parameter provided.

- SetKey shall return a SecurityErrorDomain::kUsageViolation error, if kAllowSignature is not set.
- SetKey shall return a SecurityErrorDomain::kIncompatibleObject error, if the AlgId of the provided key object is incompatible with the AlgId specified during creation of this SignerPrivateCtx.

⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_01821]{DRAFT} ⌈The SetKey interface of the VerifierPublicCtx shall check the allowed-usage flags of the key parameter provided.

- SetKey shall return a SecurityErrorDomain::kUsageViolation error, if kAllowVerification is not set.
- SetKey shall return a SecurityErrorDomain::kIncompatibleObject error, if the AlgId of the provided key object is incompatible with the AlgId specified during creation of this VerifierPublicCtx.

⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_01822]{DRAFT} ⌈The SetKey interface of the SigEncodePrivateCtx shall check the allowed-usage flags of the key parameter provided.

- SetKey shall return a SecurityErrorDomain::kUsageViolation error, if kAllowSignature is not set.
- SetKey shall return a SecurityErrorDomain::kIncompatibleObject error, if the AlgId of the provided key object is incompatible with the AlgId specified during creation of this SigEncodePrivateCtx.

⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_01823]{DRAFT} ⌈The SetKey interface of the MsgRecoveryPublicCtx shall check the allowed-usage flags of the key parameter provided.

- SetKey shall return a SecurityErrorDomain::kUsageViolation error, if kAllowVerification is not set.
- SetKey shall return a SecurityErrorDomain::kIncompatibleObject error, if the AlgId of the provided key object is incompatible with the AlgId specified during creation of this MsgRecoveryPublicCtx.

⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_02415]{DRAFT} **Pre-hashed signing** ⌈The functions SignPreHashed shall implement the signing algorithm configured for this context without hashing. Note: hashing has already been applied by the user.

- SignPreHashed shall return a SecurityErrorDomain::kProcessingNotFinished error, if a HashFunctionCtx has been supplied and the hash value computation has not been finished.
- SignPreHashed shall return a SecurityErrorDomain::kUninitializedContext error, if the SetKey was not called before.
- SignPreHashed shall return a SecurityErrorDomain::kInvalidInputSize error, if a supplied ReadOnlyMemRegion parameter's size is incompatible with the configured signature algorithm.
- SignPreHashed shall return a SecurityErrorDomain::kInvalidArgument error, if the AlgId of the provided HashFunctionCtx or the directly provided AlgId is incompatible with the configured signature algorithm.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02416]{DRAFT} **Signing** ⌈The function Sign shall implement the signing algorithm configured for this context.

- Sign shall return a SecurityErrorDomain::kUninitializedContext error, if the SetKey was not called before.
- Sign shall return a SecurityErrorDomain::kInvalidInputSize error, if a supplied ReadOnlyMemRegion parameter's size is incompatible with the configured signature algorithm.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02417]{DRAFT} **Pre-hashed verification** ⌈The functions VerifyPreHashed shall implement the verification algorithm configured for this context without hashing. Note: hashing has already been applied by the user.

- VerifyPreHashed shall return a SecurityErrorDomain::kProcessingNotFinished error, if a HashFunctionCtx has been supplied and the hash value computation has not been finished.
- VerifyPreHashed shall return a SecurityErrorDomain::kUninitializedContext error, if the SetKey was not called before.

- `VerifyPreHashed` shall return a `SecurityErrorDomain::kInvalidInputSize` error, if a supplied `ReadOnlyMemRegion` parameter's size is incompatible with the configured signature algorithm.
- `VerifyPreHashed` shall return a `SecurityErrorDomain::kInvalidArgument` error, if the `AlgId` of the provided `HashFunctionCtx` or the directly provided `AlgId` is incompatible with the configured signature algorithm.
- `VerifyPreHashed` shall return a `SecurityErrorDomain::kIncompatibleObject` error, if the `CryptoAlgId` of this context does not match the `CryptoAlgId` of signature; or the required `CryptoAlgId` of the hash is not `kAlgIdDefault` and the required hash `CryptoAlgId` of this context does not match `hashAlgId` or the hash `CryptoAlgId` of signature.
- `VerifyPreHashed` shall return a `SecurityErrorDomain::kIncompatibleArguments` error, if the provided hash `AlgId` is not `kAlgIdDefault` and the `AlgId` of the provided signature object does not match the provided hash `AlgId`.
- `VerifyPreHashed` shall return a `SecurityErrorDomain::kBadObjectReference` error, if the provided signature object does not reference the public key loaded to the context, i.e. if the COUID of the public key in the context is not equal to the COUID referenced from the signature object.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02418]{DRAFT} **Truncation of hash value** ⌈The functions `VerifyPreHashed` and `SignPreHashed` shall truncate the provided hash value, if the bitlength of the provided hash value is larger than bitlength used for signing/verification and if the applied algorithm according to the `AlgId` allows the use of a hash-value with the provided bitlength and specifies a truncation.⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02419]{DRAFT} **Signing** ⌈The function `Verify` shall implement the verification algorithm configured for this context.

- `Verify` shall return a `SecurityErrorDomain::kUninitializedContext` error, if the `SetKey` was not called before.
- `Verify` shall return a `SecurityErrorDomain::kInvalidInputSize` error, if a supplied `ReadOnlyMemRegion` parameter's size is incompatible with the configured signature algorithm.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02403]{DRAFT} ⌈The function `SignAndEncode` shall calculate a signature and return a signature that includes the encoded message according to the configured context `AlgId`.⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02421]{DRAFT} ⌈The function `DecodeAndVerify` shall decode the message from the provided signature and return the message after successful verification according to the configured context `AlgId`.⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02420]{DRAFT} ⌈The function `SignAndEncode` shall implement the sign and encode algorithm configured for this context.

- `SignAndEncode` shall return a `SecurityErrorDomain::kIncorrectInputSize` error, if the provided message data is larger than allowed by the configured context `AlgId`.
- `SignAndEncode` shall return a `SecurityErrorDomain::kUninitializedContext` error, if `SetKey` has not been called before.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_02422]{DRAFT} ⌈The function `DecodeAndVerify` shall implement the decode and verify algorithm configured for this context.

- `DecodeAndVerify` shall return a `SecurityErrorDomain::kIncorrectInputSize` error, if the provided signature data is incomplete. Note: the configured context `AlgId` expects more data than provided.
- `DecodeAndVerify` shall return a `SecurityErrorDomain::kUninitializedContext` error, if `SetKey` has not been called before.

⌋*(RS_CRYPTO_02204)*

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

7.4.1.9 Asymmetric encryption

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

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

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

[SWS_CRYPT_02701]{DRAFT} **Creation of DecryptorPrivateCtx and EncryptorPublicCtx** ⌈Calling the function `CreateDecryptorPrivateCtx` of a `Crypto Provider` shall return an instance of `DecryptorPrivateCtx` identified by the parameter `AlgId`. Calling the function `CreateEncryptorPublicCtx` of a `Crypto`

Provider shall return an instance of EncryptorPublicCtx identified by the parameter AlgId.⌋*(RS_CRYPTO_02202)*

The AlgId is the implementation specific identifier that represents the algorithm name, as described in chapter 7.5. With this identifier the context is setup matching the asymmetric algorithm. Here, the setup can influence the organization of the cryptographic material, the provided internal buffers for keys, input, or output data and the buffers length. Some asymmetric cryptographic algorithms need specific initialization parameters. All the specific needs of an asymmetric algorithm, the corresponding standards gives detailed insights how to setup internally the Crypto Provider and its supported cryptographic primitives.

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

[SWS_CRYPT_02702]{DRAFT} ⌈The SetKey interface of the EncryptorPublicCtx shall check the allowed-usage flags of the key parameter provided. If kAllowDataEncryption is not set, a SecurityErrorDomain::kUsageViolation error shall be returned.⌋*(RS_CRYPTO_02202)*

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

[SWS_CRYPT_02704]{DRAFT} **Encrypting** ⌈The interface ProcessBlock of EncryptorPublicCtx shall execute the encryption operation using the deployed public key.⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_02705]{DRAFT} **Decrypting** ⌈The interface ProcessBlock of DecryptorPrivateCtx shall execute the decryption operation using the deployed public key.⌋*(RS_CRYPTO_02202)*

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

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

[SWS_CRYPT_02726]{DRAFT} **Errors of ProcessBlock** ⌈

- ProcessBlock shall return an SecurityErrorDomain::kUninitializedContext error, if SetKey was not called before.

- ProcessBlock shall return an SecurityErrorDomain::kIncorrectInputSize error, if suppressPadding is set to TRUE and the user provided insufficient data.

⌋*(RS_CRYPTO_02202)*

7.4.1.10 Key Encapsulation Mechanism (KEM)

Briefly, a key encapsulation mechanism (KEM) works just like a public-key encryption scheme, except that the encryption algorithm takes no input other than another key. Therefore, the KEM uses randomly generated key material, the key encryption key (KEK), to encapsulate an input, in this situation a key. The input is encapsulated with an encryption with a target public key, as given in [25], [26], and [27]. The KEK can be derived from the encapsulated key material by application of a KDF. The FC Crypto support this by providing three basic building blocks:

- A key generation algorithm
- An encryption algorithm
- A decryption algorithm

The key encapsulation mechanism also needs a positive integer, which specifies the length of the output key.

[SWS_CRYPT_03000]{DRAFT} ⌈Creating a public key context for KEM is handled by the CryptoAPI. This context needs an identifier of the target KEM cryptographic algorithm.⌋*(RS_CRYPTO_02209)*

The application or functional cluster can directly use the resulting key or if needed hand over to the Key Storage Provider, which then manages the key as described in chapter 7.4.2.3.

[SWS_CRYPT_03002]{DRAFT} ⌈When the encapsulation uses a salt value to improve randomness, then the salt shall be unique for each instance of the target key.⌋ *(RS_CRYPTO_02209)*

[SWS_CRYPT_03003]{DRAFT} ⌈The FC Crypto shall allow to encapsulate a key without creating an intermediate secret seed object. However, encapsulation shall also be possible by using an intermediate secret seed object. This seed object shall improve the randomness and thus increase the secrecy of the encapsulation.⌋*(RS_CRYPTO_02209)*

[SWS_CRYPT_03004]{DRAFT} **Decryption or Decapsulation** ⌈The context shall support the reverse operation. In this operation, which is provided by the CryptoAPI, an application or functional cluster provides an encapsulated secret and the FC Crypto returns the inherent secret as a result of a cryptographic calculation. In order to execute this operation the context needs the ciphertext and the matching key material.⌋*(RS_CRYPTO_02209)*

[SWS_CRYPT_03005]{DRAFT} ⌈Encapsulating and decapsulating can be done in one context or two separate contexts. The first one allows to reuse instances on the same system and increases flexibility. The second one allows to do encapsulation and decapsulation independent from each other. This is useful when an application on a system only needs one of the operation, but not the other one. The `FC Crypto` shall support the decoupling of encapsulating and decapsulating.⌋*(RS_CRYPTO_02209)*

[SWS_CRYPT_03006]{DRAFT} **Key generation or setting** ⌈The `CryptoAPI` shall provide an interface to allow this, which allow the context to set (deploy) a key matching the specified context.⌋*(RS_CRYPTO_02209)*

This key can be either generated before it is used or the key is already known in the system and managed by the `Key Storage Provider`.

[SWS_CRYPT_03007]{DRAFT} ⌈The `CryptoAPI` shall provide an interface, which allows the context to get the entropy in bit-length of the `KEK`.⌋*(RS_CRYPTO_02209)*

[SWS_CRYPT_03008]{DRAFT} **Prefix property** ⌈The `KEM` context shall satisfy the prefix code, prefix-free codes, prefix condition codes, and instantaneous codes. This means, that all possible outputs of the encryption algorithm in the `FC Crypto` is not a prefix (initial segment) of any other byte string.⌋*(RS_CRYPTO_02209)*

[SWS_CRYPT_03009]{DRAFT} **Signalization of error** ⌈By conventions, if any algorithm fails the `FC Crypto` shall provide a distinct error. Beside the algorithm specific errors, the context will provide these errors:

- If the context was not initialized by a public key value, `SecurityErrorDomain::kUninitializedContext` shall be returned.
- If the chosen algorithm is incorrect or cannot be used, `SecurityErrorDomain::kInvalidArgument` shall be returned.
- If the output buffer contains not enough space to save the encapsulated result, `SecurityErrorDomain::kInsufficientCapacity` shall be returned.
- If the crypto primitive of the provided key object is incompatible with this symmetric key context, `SecurityErrorDomain::kIncompatibleObject` shall be returned.
- If the transformation type associated with this context is prohibited by restrictions of provided key object, `SecurityErrorDomain::kUsageViolation` shall be returned.

⌋*(RS_CRYPTO_02209)*

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

Key materials are essential elements of cryptographic algorithm. Therefore, key materials are stored locally to keep them secret. This avoids exposure and missuses.

However, there are situations when key material is exchanged, especially when another system needs to decrypt a common secret. Allowing to share key material, a secure mechanism is needed to exchange the material. Typically, this is done by asymmetric key encapsulation or key exchange mechanism (KEM). The Diffie-Hellman key exchange scheme [28] is the common used key exchange mechanism and a good example. In order to support these mechanism, the FC Crypto must provide three basic building blocks:

- A key generation algorithm
- An encryption algorithm
- A decryption algorithm

[SWS_CRYPT_03300]{DRAFT} **Key generation algorithm** ⌈For exchanging a common secret, the FC Crypto shall support the generation of a public-private key pair.⌋ *(RS_CRYPTO_02100)*

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

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

[SWS_CRYPT_03312]{DRAFT} **Decryption algorithm** ⌈The FC Crypto shall provide an encryption algorithm, which matches the chosen public-private key pair and the key exchange schema.⌋ *(RS_CRYPTO_02102)*

The matching cryptographic primitives are provided by the Crypto Provider.

[SWS_CRYPT_03313]{DRAFT} ⌈The FC Crypto shall support hybrid cryptographic system. A hybrid cryptographic system can be constructed using any two separate cryptographic primitive:

- a key encapsulation scheme, which is a public-key cryptographic primitive, and
- a data encapsulation scheme, which is a symmetric-key cryptographic primitive.

⌋ *(RS_CRYPTO_02103)*

[SWS_CRYPT_03301]{DRAFT} ⌈The FC Crypto shall allow to exchange the public key between two parties. Therefore, the FC Crypto provides an exporting of the public key.⌋ *(RS_CRYPTO_02104)*

In order to export the key, the FC Crypto offers common used key exchange mechanism, as specified in 7.4.1.10.

[SWS_CRYPT_03302]{DRAFT} ⌈Depending on the used key exchange scheme the FC Crypto shall allow encryption of arbitrary length messages and allow encryption of short messages. The length of the processed message depends on the used cryptographic primitive. The FC Crypto has to deal with arbitrary and variable length. When the input length exceeds the supported length of the cryptographic primitive, the FC

Crypto shall provide an own hybrid scheme, which deals with the encryption of long messages, or return an error to the application or functional cluster.⌋*(RS_CRYPTO_02105)*

The error indicates, that the caller has to provide an own hybrid scheme. However, most of higher-level applications or functional clusters build their own hybrid schemes, such as IKE. In such a case the FC Crypto ensures that the needed cryptographic primitives are available to build the hybrid scheme.

Another possible solution to solve the problem of arbitrarily long messages is to allow stream processing.

[SWS_CRYPT_03303]{DRAFT} **Stream processing** ⌈The FC Crypto shall support stream processing. Here, the message is presented to both encryption and decryption as an input stream. The result of the calculation will be written to an output stream. The algorithm should never have to rewind these streams.⌋*(RS_CRYPTO_02106)*

[SWS_CRYPT_03304]{DRAFT} ⌈The FC Crypto shall support to bind additional data non-malleable to the Ciphertext.⌋*(RS_CRYPTO_02107)*

[SWS_CRYPT_03305]{DRAFT} ⌈Depending on the used key exchange scheme the FC Crypto shall allow for messages and Ciphertext to be efficiently processed as "streams".⌋*(RS_CRYPTO_02108)*

The most problem during KEM is to avoid that an adversary, who is allowed to use chosen-plaintext and chosen-ciphertext attacks, can break the secrecy of a channel. Therefore, different variants of KEM were designed. Some of these uses the "provable" security scheme, which is mechanism against adaptive chosen ciphertext attack. Other schemes rely on the "random oracle" heuristic.

[SWS_CRYPT_03306]{DRAFT} ⌈The FC Crypto shall support both variants by providing the specific cryptographic algorithm.⌋*(RS_CRYPTO_02109)*

The standard does not define the variants as they are project specific.

[SWS_CRYPT_03307]{DRAFT} ⌈The FC Crypto shall support protocols that achieve forward secrecy by providing the functionality to generate new key pairs for each session and discard them at the end of the session.⌋*(RS_CRYPTO_02110)*

Some key exchange schemes allow certain types of scheme specific options to be passed to the encryption algorithm, which is allowed for an extra argument options in CryptoAPI. Allowing scheme specific options in an abstract interface is clearly not such a good idea, as this runs counter to the very notion of an abstract interface. Also, since such options are scheme specific, their use will almost certainly atrophy over time, especially if more applications take advantage of the benefits provided by an abstract encryption interface. Some elliptic curves based scheme allows the encryptor to dynamically choose, from one of several formats, how it wants to format a point on the curve.

[SWS_CRYPT_03308]{DRAFT} **Support of extra option** ⌈The FC Crypto shall support extra options based on the provided key exchange scheme.⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_03309]{DRAFT} **Signalization of error** ⌈By conventions, if any algorithm fails the FC Crypto shall provide a distinct error.⌋*(RS_CRYPTO_02112)*

[SWS_CRYPT_03310]{DRAFT} **Failing streams** ⌈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.⌋*(RS_CRYPTO_02113)*

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_03900]{DRAFT} **Translation of common name to vendor identifier** ⌈The CryptoProvider::ConvertToAlgId shall convert a common name of the cryptographic algorithm to a correspondent vendor specific algorithm identifier.⌋*(RS_CRYPTO_02308)*

[SWS_CRYPT_03902]{DRAFT} **Reverse operation** ⌈The CryptoProvider::ConvertToAlgName shall convert a vendor specific algorithm identifier to the common name of the cryptographic algorithm.⌋*(RS_CRYPTO_02308)*

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

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

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

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

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

[SWS_CRYPT_04200]{DRAFT} **Loading cryptographic material** ⌈The load interface, `CryptoProvider::LoadObject`, `CryptoProvider::LoadSymmetricKey`, `CryptoProvider::LoadPublicKey`, `CryptoProvider::LoadPrivateKey` and `CryptoProvider::LoadSecretSeed`, shall load a cryptographic object from the storage location indicated by the provided `IOInterface`.⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112, RS_CRYPTO_02113)*

[SWS_CRYPT_04201]{DRAFT} ⌈The load interface shall return

- a `SecurityErrorDomain::kModifiedResource` error, if the underlying resource has been modified after the `IOInterface` has been opened, i.e., the `IOInterface` has been invalidated.
- a `SecurityErrorDomain::kIncompatibleObject` error, if the provided `IOInterface` points to a storage location belonging to another `CryptoProvider` or the type of the storage content is not compatible to the requested return type.
- a `SecurityErrorDomain::kEmptyContainer` error, if the `IOInterface` points to an empty storage location.
- a `SecurityErrorDomain::kResourceFault` error, if the storage content is corrupted.

⌋*(RS_CRYPTO_02006)*

This is used by the `Crypto Provider` to handle key-material, which is instantiated as a persistent or volatile key slot.

[SWS_CRYPT_04212]{DRAFT} ⌈The `CryptoObject::CryptoPrimitiveId` shall return the `CryptoPrimitiveId` of the current `CryptoObject`. This information can be used by the application to identify the transformations compatible with this `CryptoObject` or check compatibility with other `CryptoObject`s.⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_04202]{DRAFT} **Exporting secure objects** ⌈The `CryptoProvider::ExportSecuredObject` shall securely wrap a `CryptoObject` using the provided `SymmetricKeyWrapperCtx`. It shall return the wrapped `CryptoObject`.⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_04213]{DRAFT} ⌈The `CryptoProvider::ExportSecuredObject` shall return

- a `SecurityErrorDomain::kIncompleteArgState` error, if the provided `SymmetricKeyWrapperCtx` is not initialized.

- a SecurityErrorDomain::kIncompatibleObject error, if the CryptoObject::IsExportable() returns false; or the key deployed to the provided SymmetricKeyWrapperCtx does not have the allowed usage flag kAllowKeyExporting set.

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_04203]{DRAFT} **Exporting public objects** ⌈The CryptoAPI shall provide an interface for exporting a public cryptographic object.⌋*(RS_CRYPTO_02004)*

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

[SWS_CRYPT_04204]{DRAFT} **Importing secure objects** ⌈The CryptoAPI shall support the reverse operation and import securely serialized objects.⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_04205]{DRAFT} **Importing public objects** ⌈The CryptoAPI shall support the reverse operation and import public serialized objects.⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_04206]{DRAFT} ⌈The CryptoAPI may execute control of an actual type of imported object and terminate the import operation at an early stage.⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_04207]{DRAFT} ⌈The CryptoAPI shall provide an interface for retrieving serialization size of all supported cryptographic objects.⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_04208]{DRAFT} ⌈The CryptoAPI shall provide an interface for creation of volatile storage element.⌋*(RS_CRYPTO_02004)*

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

[SWS_CRYPT_04209]{DRAFT} ⌈The CryptoAPI shall document all importing or exporting by a logging mechanism. This information can be queried.⌋*(RS_CRYPTO_02004)*

7.4.2 Key Storage Provider

The Key Storage Provider (KSP, namespace ara::crypto::keys) is responsible for secure (confidential and or authentic) storage of different type key material (public, private, secret keys, or seeds) and other security critical cryptographic objects (digital signatures, hash, MAC HMAC tags). These cryptographic objects are represented as a KeySlots.

KeySlots used by application are defined by the integrator in the manifest via CryptoKeySlot.

CryptoKeySlotInterface and CryptoKeySlotToPortPrototypeMapping

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

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.

[SWS_CRYPT_10003]{DRAFT} ⌈The CryptoAPI shall provide a function to obtain CryptoProvider. With a call of MyProvider the FC Crypto provides the correspinding CryptoProvider of a KeySlot.⌋*(RS_CRYPTO_02009)*

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

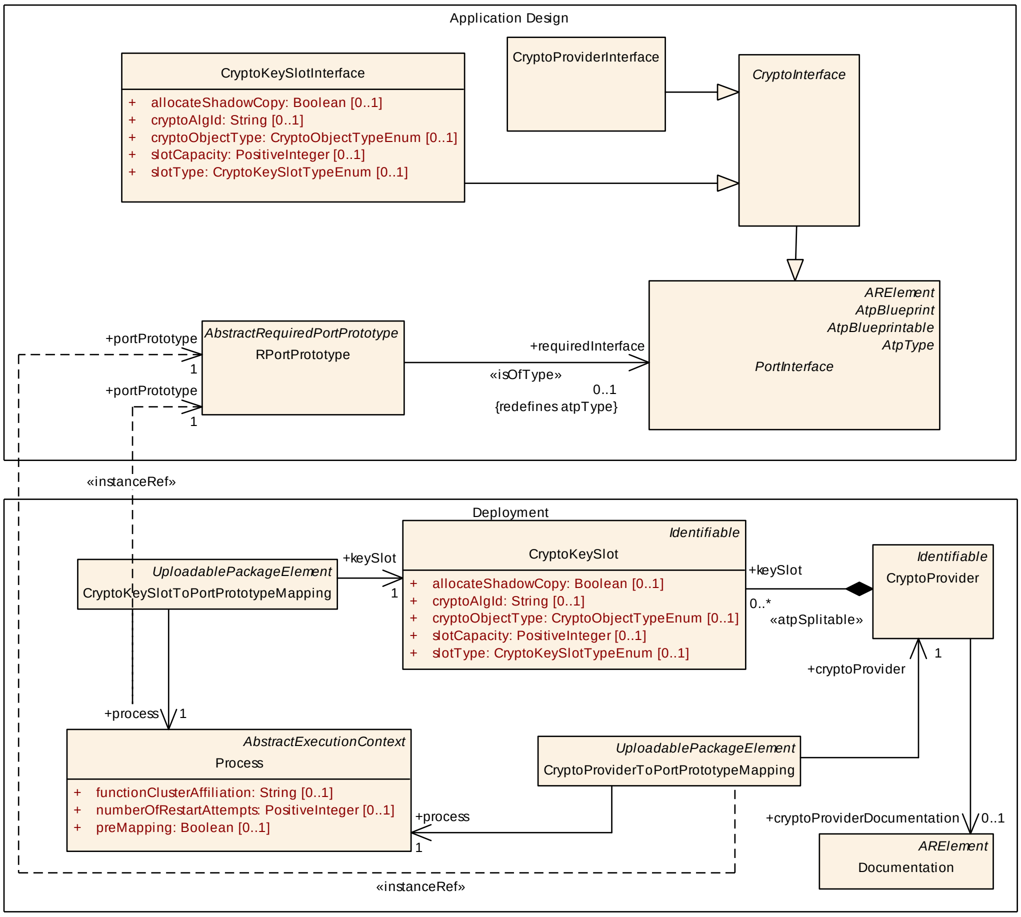

Figure 7.7: Key deployment

[SWS_CRYPT_10005]{DRAFT} ⌈The KeySlot shall be identified during runtime. The CryptoAPI provides an interface with a call to LoadKeySlot to support this. The interface needs an InstanceSpecifier as an input parameter. Here, InstanceSpecifier represents a path to RPortPrototype mapped to needed CryptoKeySlot.⌋*(RS_CRYPTO_02405)*

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

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

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::Crypto Provider interface). These methods represent all actions that need implementation of cryptographic transformations of keys. The usage of HSM is implemented in hardware and thus may not support all APIs as software solutions would.
4. Generic serialization of public cryptographic objects (via crypto::Serializable interface). Taking into account the deep dependence of 3rd category of the "Key Management" sub-API from other cryptographic functionality, possibility to reuse some functional blocks (including mechanisms of access control to key material in HSM realms), there is no practical sense to separate this sub-API from Crypto Provider API.

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

7.4.2.1 Serializable interface

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

Interfaces of all public (non-confidential) cryptographic objects and certificates that principally support serialization in plain (non-encrypted and non-authenticated) form are derived from the crypto::Serializable interface. Actually, this interface provides only one serialization method ExportPublicly.

7.4.2.2 Key Generation

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

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

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

   [SWS_CRYPT_10301]{DRAFT} ⌈The FC Crypto shall allocate a new symmetric key object by a call of function GenerateSymmetricKey and fill it by a new randomly generated value.

   - GenerateSymmetricKey shall return a SecurityErrorDomain::kUnknownIdentifier error, if the AlgId has an unsupported value.
   - GenerateSymmetricKey shall return a SecurityErrorDomain::kIncompatibleArguments error, if allowedUsage is incompatible with target algorithm specified by AlgId.

   ⌋*(RS_CRYPTO_02101)*.

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

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

   [SWS_CRYPT_10304]{DRAFT} ⌈The FC Crypto shall support the asymmetric key generation. The CrypotAPI provide such functionality. The private key is generated by calling GeneratePrivateKey.⌋*(RS_CRYPTO_02101)*

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

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

7.4.2.3 Exporting and Importing of Key Material

Exporting of key material is sometimes necessary. This is useful during the setup of communication channels, for example. Importing key material is also important for a later use. Export and Import facilities of CryptoProvider 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, KeyAgreementPrivateCtx and 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 HashFunctionCtx and SignerPrivateCtx. All contexts contains two building blocks:

- The encryption algorithm
- The decryption algorithm

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

[SWS_CRYPT_10401]{DRAFT} ⌈Key agreement private context shall provide functionality to produce a common secret seed.⌋*(RS_CRYPTO_02007)*

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

7.4.3 Certificate handling (X.509 Provider)

X.509 Certificate Management Provider (X.509 Provider) is responsible for X.509 certificates parsing, verification, authentic storage and local searching by different attributes. In addition, X.509 Provider is responsible for storage, management, and processing of Certificate Revocation Lists (CRLs) and Delta CRLs. The X.509 Provider supports the preparation of requests, responses, and parsing according to the Online Certificate Status Protocol (OCSP) as defined in [29] and [30].

[SWS_CRYPT_20000]{DRAFT} ⌈FC Crypto supports only a single instance of the X.509 Provider. As the X.509 Provider is completely independent from CryptoProvider and KeyStorageProvider implementation details, it is possible that different vendors provide X.509 Provider and CryptoProvider / KeyStorageProvider. Therefore, the standardized CryptoAPI guarantees interoperability between these independent building blocks. Applications or functional clusters can access certificates by CryptoCertificateInterface, which is provided by X.509 Provider.⌋*(RS_CRYPTO_02307)*

Any FC Crypto implementation shall include a single X.509 Provider. Responsibility of this provider is the support of Public Key Infrastructure (PKI) as defined in [31]. A PKI contains a root certificate and one or many certificates. Main feature are:

1. Storages of certificates, certification signing requests (CSRs), and certificate revocation lists (CRLs).
2. Complete parsing of X.509 certificates and certificate signing requests (CSR).
3. Encoding of all public components of certificate signing requests (e.g. Distinguished Names and X.509 Extensions).
4. Verification of certificates and certification chains (according to current set of trusted certificates).
5. Trust management of the stored certificates.
6. Search of certificates in local storage based on different parameters.
7. Automatic building of the trust chains according to saved certificates, CRLs, and trust configuration.

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

[SWS_CRYPT_20002]{DRAFT} ⌈The X.509 Provider shall store and provide the root 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 Key Storage Provider. All elements, which are relevant for the certification path, shall be stored with local access either hard-coded into the software or in a persistent and tamper-proof manner. The decision how to store the elements is based on:

- Updatability of certificates: When certificates shall be exchangeable or revocable, then these are stored in a volatile but persistent storage. Fixed certificates, which stay forever for example, can be stored hard-coded.
- Use case specific: An application or functional cluster can have pre-configured certificates, which are stored along side the configuration, e.g. in ARXML.
- Project specific

⌋*(RS_CRYPTO_02306)*

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

[SWS_CRYPT_20004]{DRAFT} ⌈The X.509 Provider shall support ASN.1 parsing. Thus it provides an ASN.1 parser to read the specific syntax of X.509 certificates. Typical X.509 certificates must follow the definition given in [31, X.509] and [32, RFC 5280]:

1. Certificate
    (a) Version Number
    (b) Serial Number
    (c) Signature Algorithm ID
    (d) Issuer Name
    (e) Validity period
        i. Not Before
        ii. Not After
    (f) Subject name
    (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)
        i. ...
2. Certificate Signature Algorithm
3. Certificate Signature

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

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

[SWS_CRYPT_20005]{DRAFT} **Freedom of interference during update** ⌈It must be possible to regularly update any key pair of certificates, which are part of a PKI tree, without affecting any other key pair of related certificates, which can be also part of the same PKI tree or part of an independent tree.⌋*(RS_CRYPTO_02112, RS_CRYPTO_02306)*

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

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

1. ”Persistent” storage is dedicated for X.509 artifacts that should survive after ECU restart / shutdown.
2. ”Volatile” (or ”Session”) is dedicated for X.509 artifacts, that are valuable only in scope of current session of an application or functional cluster, importing these artifacts to the storage.

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

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

[SWS_CRYPT_20010]{DRAFT} ⌈X.509 Provider implementation must require especial capability ”Trust Master” from applications that will set specific certificate as a root of trust.⌋*(RS_CRYPTO_02306)*

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

7.4.3.1 Certificate Signing Request

[SWS_CRYPT_20301]{DRAFT} ⌈The X.509 Provider produces the Certificate Signing Request (CSR). 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 CSR is signed by the private key and contains the public key.⌋ *(RS_CRYPTO_02306)* The identification of the used algorithm is done by the common name, as specified in 7.5.

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

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

[SWS_CRYPT_20303]{DRAFT} ⌈All meta-information shall be encoded according to the X.509 specification (as given in [31], [33], [34], [35], [36], [37], and [2]).⌋ *(RS_CRYPTO_02306)*

X.509 Provider distinguishes three states of a CSR:

1. "New" - the CSR is created, but is not yet sent to the Certification Authority (CA).
2. "Pending" - the CSR was already sent to the CA, but the internal was not yet updated. Either the CSR was not returned or was not processed.
3. "Retrieved" - the CSR was returned from the CA, and is either processed or the processing was not started yet.

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

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

7.4.3.2 Using Certificates

[SWS_CRYPT_20601]{DRAFT} **Importing / Installation** ⌈The X.509 Provider provides a mechanism for applications or functional clusters to import or install certificates, parts of certification paths, or full certification paths.⌋ *(RS_CRYPTO_02306)*

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

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

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

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

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

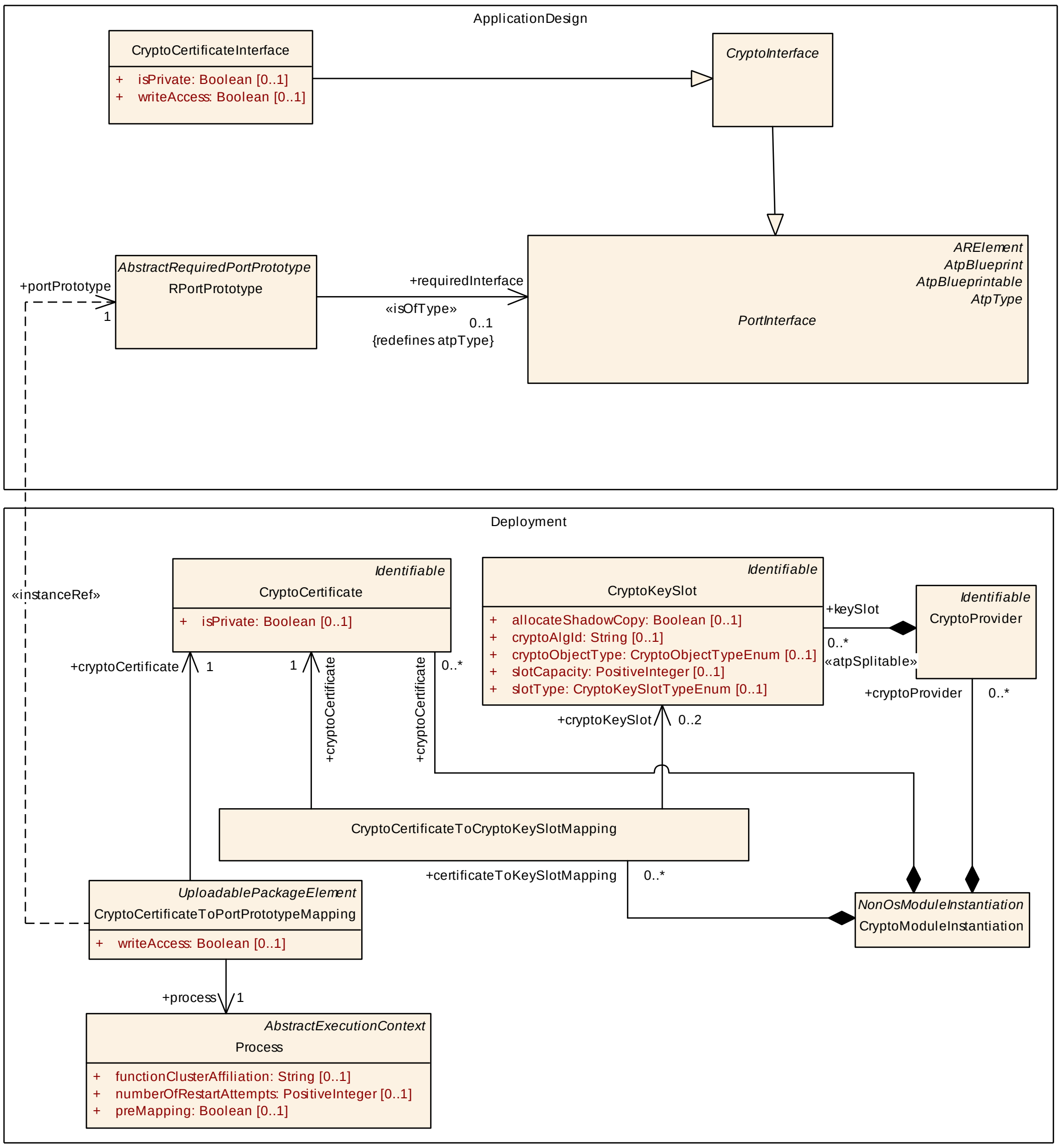

Figure 7.8: Certificate deployment

[SWS_CRYPT_20611]{DRAFT} **Valdiation of certification path** ⌈When a certificate is installed, the whole certificate chain must be validated based on the whole tree path up to the root certificate (e.g., vehicle root). Only certificates, which are not root certificates, are checked.⌋*(RS_CRYPTO_02306)*

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

[SWS_CRYPT_20612]{DRAFT} ⌈Supporting a full certificate life-cycle, the FC Crypto provides functionality to generate certificate signing request, where the needed encoding (i.e., DER or PEM) can be specified and the correct setting is ensured. The CryptoAPI provides this interface for CSR generation. Additionally, the

CryptoAPI offers the specific interfaces to generate certificates and certificate chains, which can then be used by other protocols, i.e., IKE.⌋(RS_CRYPTO_02306)

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

[SWS_CRYPT_20613]{DRAFT} ⌈The FC Crypto allows to encode and decode ASN.1-based standard formats (like [38, PKCS#8], [39, PKCS#12]), as specified in [40, X.680], [41, X.682], and [42, X.683]. The CryptoAPI allows an application or functional cluster to select the encoding.⌋*(RS_CRYPTO_02306)*

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

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

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

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

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

[SWS_CRYPT_20619]{DRAFT} ⌈During the initialization of the FC Crypto, all needed steps for service instantiation is done. This includes importing a root CA public key, setting up the certification path with all public keys along the path, checking the revocation status of certificates, updating the X.509 Provider internal management structure with certificate status, and the certificate ecosystem.⌋*(RS_CRYPTO_02306)*

7.4.3.3 Revocation of certificates

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

[SWS_CRYPT_20901]{DRAFT} **CRL and CTL usage** ⌈The X.509 Provider shall support CRL and CTL. The format of CRL and CTL are defined in [32, RFC 5280], [43, RFC 6518], [44, RFC 8398], and [45, RFC 8399] and is not part of this standard. The X.509 Provider can store the CRL and the CTL in an own internal used structure. However, the X.509 Provider can also use the provided information to update the corresponding elements. The update can be either the deletion of the element or setting a mark that the element was revoked.⌋*(RS_CRYPTO_02306)*

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

[SWS_CRYPT_20902]{DRAFT} ⌈Given in [32] the CRL can contain two different states:

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

⌋*(RS_CRYPTO_02306)*

CryptoAPI shall provide two ways to get CRL:

1. Offline: An application or functional cluster provides a CRL to the X.509 Provider.
2. Online: X.509 Provider opens a secure channel to a backend system. After a successful established connection, the X.509 Provider gets the matching CRL. The location of the specific backend system can either configured or provided via an application or functional cluster.

[SWS_CRYPT_20903]{DRAFT} **Import** ⌈The X.509 Provider allows to import 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 ImportCrl, which is provided by the CryptoAPI.⌋*(RS_CRYPTO_02306)*

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

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

- In offline mode: An application or functional cluster provides the CRL to the X.-509 Provider via an interface, which is exposed by the CryptoAPI.
- in online mode: The X.509 Provider uses a provided location to get the CRL. The location was provided by configuration or given in the interface call.

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

[SWS_CRYPT_20906]{DRAFT} ⌈The X.509 Provider shall support the standard protocol, OCSP (as defined in [30, RFC 6960]) and OCSP Stapling (as defined in [46, RFC 6066], [47, RFC 6961], and [48, RFC 8446]), to check if a certificate is revoked. OCSP is an alternative to CRLs.⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_20907]{DRAFT} ⌈The CryptoAPI provides a method to generate an OCSP request, which is defined in [30, RFC 6960]. The method can be used by an application or functional cluster.⌋*(RS_CRYPTO_02306)*

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

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

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

7.5 Cryptographic Primitives Naming Convention

CryptoProviders 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 `CryptoProvider`.
- They serve as identifiers to cryptographic algorithms.
- The `CryptoProvider` interprets the string names and matches it to the algorithm, which is provided by `FC Crypto`.

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

1. The string names contains only Latin alphanumeric characters.
2. The string names contain up to 6 delimiters for cryptographic algorithm definition.
3. The string names is case insensitive. Thus, all comparisons of the identifiers shall be always case insensitive.
4. The string names to identify cryptographic algorithms shall satisfy the following structures:

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

   where

   - `"{TargetTransformation(Mode)}"` – a specifier of target transformation: for complex transformations it is a mode name, but for fully-defined algorithms it is just their name.
   - `"{SupportingAlgorithms}"` – a specifier of basic cryptographic algorithm(s) including key length andor block length.
   - `"{Encoding&Padding}"` – a specifier of encoding and/or padding method. It can support following predefined name (equal to empty specification):
     - `"Zero"` – a default encoding & padding method: if data are already aligned to the block boundary then it doesn't add anything, but if they are not aligned then applies a padding by `'\0'` bytes up to the block boundary.

Allowed delimiters:

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

- '_' – separator instead of general separation characters (e.g.: ' ', '.', ':', '-', '') in original name of standard. This delimiter can be applied between two digits or two letters only!
- '-' – separator between a base algorithm name and its precise specifiers that define key-length or block-length in bits.
- '+' – separator between a few base algorithms' specifications for a cascade transformation definition.
- ',' – separator between a few base algorithms' specifications for a case if the whole algorithm is based on a few types of basic transformations.
- '.' – separator between a common name of a standard and its specific part or its version that precises a specification of concrete transformation.

⌋*(RS_CRYPTO_02308)*

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

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

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

Examples of fully defined transformations:

- "ECDSA-384" means ECDSA signature algorithm with private key-length 384 bit.
- "ECDH-512" means ECDH key agreement algorithm with private key-length 512 bit.
- "CTRAES-256" means a CTR-mode stream cipher based on AES algorithm with key-length 256 bit.
- "CBCAES-192+Camellia-192/PKCS7" means CBC-mode cipher based on cascade application of AES-192 and Camellia-192 with padding of last block according to PKCS#7.
- "HMACSHA-256" means HMAC based on SHA-256.

If an algorithm support a few variable length parameters then they shall be specified in following order:
key, IO-block or output digest, IV or input block (e.g.: "Kalyna-512-256" means block cipher Kalina with 512-bit key and 256-bit block).

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

- `"SIG"` – signature primitive (e.g., `"SIGRSA-2048,SHA-160PKCS1.v1_5,EMSA"`)
- `"VER"` – verification primitive (e.g., `"VERRSA-2048,SHA-160PKCS1.v1_5,EMSA"`)
- `"ENC"` – encryption primitive (e.g., `"ENCRSA-2048,MGF1,SHA-160PKCS1.v1_5,EME"`, `"ENCRSA-4096,MGF1,SHA2-256OAEP,EME"`)
- `"DEC"` – decryption primitive (e.g., `"DECRSA-2048,MGF1,SHA-160PKCS1.v1_5,EME"`, `"DECRSA-4096,MGF1,SHA2-256OAEP,EME"`)
- `"KEM"` – Key Encapsulation Mechanism (e.g., `"KEM/RSA-2048,AES-128,KDF3,SHA-256"`)

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

8 API specification

8.1 C++ language binding Crypto Provider

[SWS_CRYPT_20100]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_29030]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_20400]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02008)*

[SWS_CRYPT_20500]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20600]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20700]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02305, RS_CRYPTO_02307, RS_CRYPTO_02401)*

[SWS_CRYPT_29020]{DRAFT} ⌈

Kind:	class
Symbol:	CryptoService
Scope:	namespace ara::crypto::cryp
Base class:	ExtensionService
Syntax:	`class CryptoService : public ExtensionService {...};`

▽

△

Header file:	#include "ara/crypto/cryp/crypto_service.h"
Description:	Extension meta-information service for cryptographic contexts.

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_20800]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_29010]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_21000]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_29040]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_21100]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02205)*

[SWS_CRYPT_21300]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02104)*

[SWS_CRYPT_21400]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02209)*

[SWS_CRYPT_21500]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02103)*

[SWS_CRYPT_21800]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02209)*

[SWS_CRYPT_22100]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_22200]{DRAFT} ⌈

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

▽

△

Description:	A public key context for asymmetric recovery of a short message and its signature verification (RSA-like). Restricted groups of trusted subscribers can use this primitive for simultaneous provisioning of confidentiality, authenticity and non-repudiation of short messages, if the public key is generated appropriately and kept in secret. If (0 == BlockCryptor::ProcessBlock(...)) then the input message-block is violated.

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204)*

[SWS_CRYPT_22500]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02002, RS_CRYPTO_02403)*

[SWS_CRYPT_22700]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_22900]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02206)*

[SWS_CRYPT_24800]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02008)*

[SWS_CRYPT_23000]{DRAFT} ⌈

Kind:	class
Symbol:	SecretSeed
Scope:	namespace ara::crypto::cryp
Base class:	RestrictedUseObject
Syntax:	`class SecretSeed : public RestrictedUseObject {...};`
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Description:	Secret Seed object interface. This object contains a raw bit sequence of specific length (without any filtering of allowed/disallowed values)! The secret seed value can be loaded only to a non-key input of a cryptographic transformation context (like IV/salt/nonce)! Bit length of the secret seed is specific to concret crypto algorithm and corresponds to maximum of its input/ output/salt block-length.

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_23200]{DRAFT} ⌈

Kind:	class
Symbol:	SigEncodePrivateCtx
Scope:	namespace ara::crypto::cryp
Base class:	CryptoContext
Syntax:	`class SigEncodePrivateCtx : public CryptoContext {...};`
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Description:	A private key context for asymmetric signature calculation and short message encoding (RSA-like). Restricted groups of trusted subscribers can use this primitive for simultaneous provisioning of confidentiality, authenticity and non-repudiation of short messages, if the public key is generated appropriately and kept in secret.

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204)*

[SWS_CRYPT_29000]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23300]{DRAFT} ⌈

Kind:	class
Symbol:	Signature
Scope:	namespace ara::crypto::cryp
Base class:	CryptoObject
Syntax:	`class Signature : public CryptoObject {...};`
Header file:	#include "ara/crypto/cryp/cryobj/signature.h"
Description:	Signature container interface This interface is applicable for keeping the Digital Signature, Hash Digest, (Hash-based) Message Authentication Code (MAC/HMAC). In case of a keyed signature (Digital Signature or MAC/HMAC) a COUID of the signature verification key can be obtained by a call of CryptoObject::HasDependence()!

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02204, RS_CRYPTO_02205)*

[SWS_CRYPT_23500]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_23600]{DRAFT} ⌈

Kind:	class
Symbol:	StreamCipherCtx
Scope:	namespace ara::crypto::cryp
Base class:	CryptoContext
Syntax:	`class StreamCipherCtx : public CryptoContext {...};`

▽

△

Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Description:	Generalized Stream Cipher Context interface (it covers all modes of operation).

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_23700]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_23800]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02403)*

[SWS_CRYPT_24000]{DRAFT} ⌈

Kind:	class
Symbol:	SymmetricKeyWrapperCtx
Scope:	namespace ara::crypto::cryp
Base class:	CryptoContext
Syntax:	`class SymmetricKeyWrapperCtx : public CryptoContext {...};`
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"
Description:	Context of a symmetric key wrap algorithm (for AES it should be compatible with RFC3394 or RFC5649). The public interface of this context is dedicated for raw key material wrapping/ unwrapping, i.e. without any meta-information assigned to the key material in source crypto object. But additionally this context type should support some "hidden" low-level methods suitable for whole crypto object exporting/importing. Key Wrapping of a whole crypto object (including associated meta-information) can be done by methods: ExportSecuredObject() and ImportSecuredObject(), but without compliance to RFC3394 or RFC5649.

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02208)*

[SWS_CRYPT_24100]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_20319]{DRAFT} ⌈

Kind:	function	
Symbol:	Check(const Signature &expected)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<bool> Check (const Signature &expected) const noexcept=0;`	
Parameters (in):	expected	the signature object containing an expected digest value
Return value:	ara::core::Result< bool >	true if value and meta-information of the provided "signature" object is identical to calculated digest and current configuration of the context respectively; but false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Finished	if the digest calculation was not finished by a call of the Finish() method
	SecurityErrorDomain::kIncompatible Object	if the provided "signature" object was produced by another crypto primitive type
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Check the calculated digest against an expected "signature" object. Entire digest value is kept in the context up to next call Start(), therefore it can be verified again or extracted. This method can be implemented as "inline" after standartization of function ara::core::memcmp().	

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02204)*

[SWS_CRYPT_20102]{DRAFT} ⌈

Kind:	function	
Symbol:	GetBlockService()	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual BlockService::Uptr GetBlockService () const noexcept=0;`	
Return value:	BlockService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Get BlockService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_20316]{DRAFT} ⌈

Kind:	function	
Symbol:	GetDigest(std::size_t offset=0)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > GetDigest (std::size_t offset=0) const noexcept;`	
Parameters (in):	offset	position of the first byte of digest that should be placed to the output buffer
Return value:	ara::core::Result< ByteVector< Alloc > >	an output buffer storing the requested digest fragment or the full digest
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Finished	if the digest calculation was not finished by a call of the Finish() method
	SecurityErrorDomain::kUsageViolation	if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context initialized by a key without kAllow Signature permission
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Retrieve the calculated digest. The entire digest value is kept in the context until the next call of Start(). Therefore, the digest can be re-checked or extracted at any time. If the offset is larger than the digest, an empty buffer shall be returned. This method can be implemented as "inline" after standardization of function ara::core::memcpy().	

⌋*(RS_CRYPTO_02207)*

[SWS_CRYPT_21715]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_20103]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_23634]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessConfidentialData(ReadOnlyMemRegion in, ReadOnlyMemRegion expectedTag=nullptr)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ProcessConfidentialData (ReadOnlyMemRegion in, ReadOnlyMemRegion expectedTag=nullptr) noexcept=0;`	
Parameters (in):	in	the input buffer containing the full message
	expectedTag	pointer to read only mem region
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if size of the input buffer is not divisible by the block size (see GetBlockSize())
	SecurityErrorDomain::kProcessingNotStarted	if the data processing was not started by a call of the Start() method
	SecurityErrorDomain::kAuthTagNotValid	if the processed data cannot be authenticated
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Process confidential data The input buffer will be overwritten by the processed message. This function is the final call, i.e. all associated data must have been already provided. Hence, the function will check the authentication tag and only return the processed data, if the tag is valid.	

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_23635]{DRAFT} ⌈

Kind:	function
Symbol:	ProcessConfidentialData(ReadWriteMemRegion inOut, ReadOnlyMemRegion expectedTag=nullptr)
Scope:	class ara::crypto::cryp::AuthCipherCtx

▽

△

Syntax:	`virtual ara::core::Result<void> ProcessConfidentialData (ReadWriteMemRegion inOut, ReadOnlyMemRegion expectedTag=nullptr) noexcept=0;`	
Parameters (in):	inOut	the input buffer containing the full message
	expectedTag	pointer to read only mem region
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if size of the input buffer is not divisible by the block size (see GetBlockSize())
	SecurityErrorDomain::kProcessingNotStarted	if the data processing was not started by a call of the Start() method
	SecurityErrorDomain::kAuthTagNotValid	if the processed data cannot be authenticated
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Process confidential data The input buffer will be overwritten by the processed message After this method is called no additional associated data may be updated.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_20414]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_23911]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt) noexcept=0;`	
Parameters (in):	key	the source key object
DIRECTION NOT DEFINED	transform	–
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Set (deploy) a key to the authenticated cipher symmetric algorithm context.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_24714]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(ReadOnlyMemRegion iv=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<void> Start (ReadOnlyMemRegion iv=ReadOnly MemRegion()) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized
	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false)
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Initialize the context for a new data processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_24715]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(const SecretSeed &iv)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<void> Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	

▽

△

Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized
	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation
	SecurityErrorDomain::kUsageViolation	if this transformation type is prohibited by the "allowed usage" restrictions of the provided Secret Seed object
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Initialize the context for a new data processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_20312]{DRAFT} ⌈

Kind:	function	
Symbol:	UpdateAssociatedData(const RestrictedUseObject &in)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<void> UpdateAssociatedData (const RestrictedUseObject &in) noexcept=0;`	
Parameters (in):	in	a part of input message that should be processed
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Started	if the digest calculation was not initiated by a call of the Start() method
	SecurityErrorDomain::kInvalidUsage Order	if ProcessConfidentialData has already been called
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Update the digest calculation by the specified RestrictedUseObject. This method is dedicated for cases then the RestrictedUseObject is a part of the "message".	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_20313]{DRAFT} ⌈

Kind:	function	
Symbol:	UpdateAssociatedData(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<void> UpdateAssociatedData (ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	a part of the input message that should be processed

▽

△

Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNotStarted	if the digest calculation was not initiated by a call of the Start() method
	SecurityErrorDomain::kInvalidUsageOrder	if ProcessConfidentialData has already been called
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Update the digest calculation by a new chunk of associated data.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_20314]{DRAFT} ⌈

Kind:	function	
Symbol:	UpdateAssociatedData(std::uint8_t in)	
Scope:	class ara::crypto::cryp::AuthCipherCtx	
Syntax:	`virtual ara::core::Result<void> UpdateAssociatedData (std::uint8_t in) noexcept=0;`	
Parameters (in):	in	a byte value that is a part of input message
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNotStarted	if the digest calculation was not initiated by a call of the Start() method
	SecurityErrorDomain::kInvalidUsageOrder	if ProcessConfidentialData has already been called
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"	
Description:	Update the digest calculation by the specified Byte. This method is convenient for processing of constant tags.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_29035]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29033]{DRAFT} ⌈

Kind:	function	
Symbol:	GetBlockSize()	
Scope:	class ara::crypto::cryp::BlockService	
Syntax:	`virtual std::size_t GetBlockSize () const noexcept=0;`	
Return value:	std::size_t	size of the block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/block_service.h"	
Description:	Get block (or internal buffer) size of the base algorithm.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29032]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29034]{DRAFT} ⌈

Kind:	function	
Symbol:	IsValidIvSize(std::size_t ivSize)	
Scope:	class ara::crypto::cryp::BlockService	
Syntax:	`virtual bool IsValidIvSize (std::size_t ivSize) const noexcept=0;`	
Parameters (in):	ivSize	the length of the IV in bytes
Return value:	bool	true if provided IV length is supported by the algorithm and false otherwise

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/block_service.h"
Description:	Verify validity of specific Initialization Vector (IV) length.

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_20401]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02008)*

[SWS_CRYPT_20411]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCryptoPrimitiveId()	
Scope:	class ara::crypto::cryp::CryptoContext	
Syntax:	`virtual CryptoPrimitiveId::Uptr GetCryptoPrimitiveId () const noexcept=0;`	
Return value:	CryptoPrimitiveId::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Description:	Return CryptoPrimitivId instance containing instance identification.	

⌋*(RS_CRYPTO_02008)*

[SWS_CRYPT_20412]{DRAFT} ⌈

Kind:	function	
Symbol:	IsInitialized()	
Scope:	class ara::crypto::cryp::CryptoContext	
Syntax:	`virtual bool IsInitialized () const noexcept=0;`	
Return value:	bool	true if the crypto context is completely initialized and ready to use, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_context.h"	

▽

△

Description:	Check if the crypto context is already initialized and ready to use. It checks all required values, including: key value, IV/seed, etc.

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_30214]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const CryptoContext &other)	
Scope:	class ara::crypto::cryp::CryptoContext	
Syntax:	`CryptoContext& operator= (const CryptoContext &other)=default;`	
Parameters (in):	other	the other instance
Return value:	CryptoContext &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Description:	Copy-assign another CryptoContext to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30215]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(CryptoContext &&other)	
Scope:	class ara::crypto::cryp::CryptoContext	
Syntax:	`CryptoContext& operator= (CryptoContext &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	CryptoContext &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Description:	Move-assign another CryptoContext to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_20654]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02401)*

[SWS_CRYPT_20503]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20518]{DRAFT} ⌈

Kind:	function	
Symbol:	Downcast(CryptoObject::Uptrc &&object)	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`template <class ConcreteObject>` `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 Crypto Object interface
Return value:	ara::core::Result< typename Concrete Object::Uptrc >	unique smart pointer to downcasted constant interface of specified derived type
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kBadObjectType	if an actual type of the object is not the specified ConcreteObject
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Downcast and move unique smart pointer from the generic CryptoObject interface to concrete derived object.	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20505]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCryptoPrimitiveId()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual CryptoPrimitiveId::Uptr GetCryptoPrimitiveId () const noexcept=0;`	
Return value:	CryptoPrimitiveId::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Return the CryptoPrimitivId of this CryptoObject.	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20514]{DRAFT} ⌈

Kind:	function	
Symbol:	GetObjectId()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual COIdentifier GetObjectId () const noexcept=0;`	
Return value:	COIdentifier	the object's COIdentifier including the object's type and COUID (or an empty COUID, if this object is not identifiable).
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Return the object's COIdentifier, which includes the object's type and UID. An object that has no assigned COUID cannot be (securely) serialized / exported or saved to a non-volatile storage. An object should not have a COUID if it is session and non-exportable simultaneously A few related objects of different types can share a single COUID (e.g. private and public keys), but a combination of COUID and object type must be unique always!	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20516]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPayloadSize()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual std::size_t GetPayloadSize () const noexcept=0;`	
Return value:	std::size_t	size in bytes of the object's payload required for its storage
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Return actual size of the object's payload. Returned value always must be less than or equal to the maximum payload size expected for this primitive and object type, it is available via call: My Provider().GetPayloadStorageSize(GetObjectType(), GetPrimitiveId()).Value(); Returned value does not take into account the object's meta-information properties, but their size is fixed and common for all crypto objects independently from their actual type. During an allocation of a TrustedContainer, Crypto Providers (and Key Storage Providers) reserve space for an object's meta-information automatically, according to their implementation details.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_20515]{DRAFT} ⌈

Kind:	function	
Symbol:	HasDependence()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual COIdentifier HasDependence () const noexcept=0;`	
Return value:	COIdentifier	target COIdentifier of the existing dependence or CryptoObjectType::kUnknown and empty COUID, if the current object does not depend on another CryptoObject

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Description:	Return the COIdentifier of the CryptoObject that this CryptoObject depends on. For signatures objects this method must return a reference to correspondent signature verification public key! Unambiguous identification of a CryptoObject requires both components: CryptoObjectUid and CryptoObjectType.

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20513]{DRAFT} ⌈

Kind:	function	
Symbol:	IsExportable()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual bool IsExportable () const noexcept=0;`	
Return value:	bool	true if the object is exportable (i.e. if it can be exported outside the trusted environment of the Crypto Provider)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Get the exportability attribute of the crypto object. An exportable object must have an assigned COUID (see GetObjectId()).	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20512]{DRAFT} ⌈

Kind:	function	
Symbol:	IsSession()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual bool IsSession () const noexcept=0;`	
Return value:	bool	true if the object is temporay (i.e. its life time is limited by the current session only)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Return the "session" (or "temporary") attribute of the object. A temporary object cannot be saved to a persistent storage location pointed to by an IOInterface! A temporary object will be securely destroyed together with this interface instance! A non-session object must have an assigned COUID (see GetObjectId()).	

⌋*(RS_CRYPTO_02003)*

[SWS_CRYPT_20517]{DRAFT} ⌈

Kind:	function	
Symbol:	Save(IOInterface &container)	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual ara::core::Result<void> Save (IOInterface &container) const noexcept=0;`	
Parameters (in):	container	IOInterface representing underlying storage
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the object is "session", but the IOInterface represents a KeySlot.
	SecurityErrorDomain::kContent Restrictions	if the object doesn't satisfy the slot restrictions (
	SecurityErrorDomain::kInsufficient Capacity	if the capacity of the target container is not enough, i.e. if (container.Capacity() < this->StorageSize())
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	SecurityErrorDomain::kUnreserved Resource	if the IOInterface is not opened writeable.
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Save itself to provided IOInterface A CryptoObject with property "session" cannot be saved in a KeySlot.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30208]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const CryptoObject &other)	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`CryptoObject& operator= (const CryptoObject &other)=default;`	
Parameters (in):	other	the other instance
Return value:	CryptoObject &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Copy-assign another CryptoObject to this instance.	

⌋*(RS_CRYPTO_02009)*

[SWS_CRYPT_30209]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(CryptoObject &&other)	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`CryptoObject& operator= (CryptoObject &&other)=default;`	
Parameters (in):	other	the other instance

▽

△

Return value:	CryptoObject &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"	
Description:	Move-assign another CryptoObject to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_10808]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_20652]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_20651]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPrimitiveName()	
Scope:	class ara::crypto::cryp::CryptoPrimitiveId	
Syntax:	`virtual const ara::core::StringView GetPrimitiveName () const noexcept=0;`	
Return value:	const ara::core::StringView	the unified name of the crypto primitive
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"	

▽

△

Description:	Get a unified name of the primitive. The crypto primitive name can be fully or partially specified (see "Crypto Primitives Naming Convention" for more details). The life-time of the returned StringView instance should not exceed the life-time of this CryptoPrimitiveId instance!

⌋(*RS_CRYPTO_02308*)

[SWS_CRYPT_30212]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const CryptoPrimitiveId &other)	
Scope:	class ara::crypto::cryp::CryptoPrimitiveId	
Syntax:	`CryptoPrimitiveId& operator= (const CryptoPrimitiveId &other)=default;`	
Parameters (in):	other	the other instance
Return value:	CryptoPrimitiveId &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"	
Description:	Copy-assign another CryptoPrimitiveId to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30213]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(CryptoPrimitiveId &&other)	
Scope:	class ara::crypto::cryp::CryptoPrimitiveId	
Syntax:	`CryptoPrimitiveId& operator= (CryptoPrimitiveId &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	CryptoPrimitiveId &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"	
Description:	Move-assign another CryptoPrimitiveId to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_20726]{DRAFT} ⌈

Kind:	function	
Symbol:	AllocVolatileContainer(std::size_t capacity=0)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<VolatileTrustedContainer::Uptr> AllocVolatileContainer (std::size_t capacity=0) noexcept=0;`	
Parameters (in):	capacity	the capacity required for this volatile trusted container (in bytes)
Return value:	ara::core::Result< VolatileTrustedContainer::Uptr >	unique smart pointer to an allocated volatile trusted container
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Allocate a Volatile (virtual) Trusted Container according to directly specified capacity. The Volatile Trusted Container can be used for execution of the import operations. Current process obtains the "Owner" rights for allocated Container. If (capacity == 0) then the capacity of the container will be selected automatically according to a maximal size of supported crypto objects. A few volatile (temporary) containers can coexist at same time without any affecting each-other.

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

[SWS_CRYPT_20727]{DRAFT} ⌈

Kind:	function	
Symbol:	AllocVolatileContainer(std::pair< AlgId, CryptoObjectType > theObjectDef)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<VolatileTrustedContainer::Uptr> 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< VolatileTrustedContainer::Uptr >	unique smart pointer to an allocated volatile trusted container
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if unsupported combination of object type and algorithm ID presents in the list
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Allocate a Volatile (virtual) Trusted Container according to indirect specification of a minimal required capacity for hosting of any listed object. The Volatile Trusted Container can be used for execution of the import operations. Current process obtains the "Owner" rights for allocated Container. Real container capacity is calculated as a maximal storage size of all listed objects.	

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

[SWS_CRYPT_20711]{DRAFT} ⌈

Kind:	function	
Symbol:	ConvertToAlgId(ara::core::StringView primitiveName)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual AlgId 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:	AlgId	vendor specific binary algorithm ID or kAlgIdUndefined if a primitive with provided name is not supported
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	

▽

△

Description:	Convert a common name of crypto algorithm to a correspondent vendor specific binary algorithm ID.

⌋(*RS_CRYPTO_02308*)

[SWS_CRYPT_20712]{DRAFT} ⌈

Kind:	function	
Symbol:	ConvertToAlgName(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<ara::core::String> ConvertToAlgName (AlgId algId) const noexcept=0;`	
Parameters (in):	algId	the vendor specific binary algorithm ID
Return value:	ara::core::Result< ara::core::String >	the common name of the crypto algorithm (see "Crypto Primitives Naming Convention" for more details)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Convert a vendor specific binary algorithm ID to a correspondent common name of the crypto algorithm.	

⌋(*RS_CRYPTO_02308*)

[SWS_CRYPT_20745]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateAuthCipherCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<AuthCipherCtx::Uptr> CreateAuthCipherCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
Return value:	ara::core::Result< AuthCipherCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if algId argument specifies a crypto algorithm different from symmetric authenticated stream cipher
	SecurityErrorDomain::kInvalid Argument	–
	SecurityErrorDomain::kUnknown Identifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric authenticated cipher context.	

⌋(*RS_CRYPTO_02207*, *RS_AP_00144*)

[SWS_CRYPT_20751]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateDecryptorPrivateCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<DecryptorPrivateCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption
	SecurityErrorDomain::kUnknownIdentifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a decryption private key context.	

⌋*(RS_CRYPTO_02202, RS_AP_00144)*

[SWS_CRYPT_20750]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateEncryptorPublicCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<EncryptorPublicCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption
	SecurityErrorDomain::kUnknownIdentifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create an encryption public key context.	

⌋*(RS_CRYPTO_02202, RS_AP_00144)*

[SWS_CRYPT_20761]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateHashDigest(AlgId hashAlgId, ReadOnlyMemRegion value)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> CreateHashDigest (AlgId hashAlgId, ReadOnlyMemRegion value) noexcept=0;`	
Parameters (in):	hashAlgId	identifier of an applied hash function crypto algorithm
	value	raw BLOB value of the hash digest
Return value:	ara::core::Result< Signature::Uptrc >	unique smart pointer to the created Signature object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if hashAlgId argument has unsupported value
	SecurityErrorDomain::kInvalid Argument	if hashAlgId argument specifies crypto algorithm different from a hash function
	SecurityErrorDomain::kInvalidInputSize	if the value argument has invalid size (i.e. incompatible with the hashAlgId argument)
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Construct Signature object from directly provided components of a hash digest.	

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006, RS_AP_00144)*

[SWS_CRYPT_20747]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02205, RS_AP_00144)*

[SWS_CRYPT_20758]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateKeyAgreementPrivateCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<KeyAgreementPrivateCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if algId argument specifies a crypto algorithm different from key-agreement
	SecurityErrorDomain::kUnknownIdentifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a key-agreement private key context.	

⌋*(RS_CRYPTO_02104, RS_AP_00144)*

[SWS_CRYPT_20753]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateKeyDecapsulatorPrivateCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<KeyDecapsulatorPrivateCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if algId argument specifies a crypto algorithm different from asymmetric KEM
	SecurityErrorDomain::kUnknownIdentifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a key-decapsulator private key context of a Key Encapsulation Mechanism (KEM).	

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02209, RS_AP_00144)*

[SWS_CRYPT_20748]{DRAFT} ⌈

Kind:	function
Symbol:	CreateKeyDerivationFunctionCtx(AlgId algId)
Scope:	class ara::crypto::cryp::CryptoProvider

▽

△

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

⌋*(RS_CRYPTO_02103, RS_AP_00144)*

[SWS_CRYPT_20752]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02209, RS_AP_00144)*

[SWS_CRYPT_20746]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateMessageAuthCodeCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<MessageAuthnCodeCtx::Uptr> CreateMessageAuthCodeCtx (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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if algId argument specifies a crypto algorithm different from symmetric message authentication code
	SecurityErrorDomain::kUnknown Identifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric message authentication code context.	

⌋*(RS_CRYPTO_02203, RS_AP_00144)*

[SWS_CRYPT_20755]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateMsgRecoveryPublicCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<MsgRecoveryPublicCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery
	SecurityErrorDomain::kUnknown Identifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a message recovery public key context.	

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204, RS_AP_00144)*

[SWS_CRYPT_20741]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateRandomGeneratorCtx(AlgId algId=kAlgIdDefault, bool initialize=true)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<RandomGeneratorCtx::Uptr> CreateRandomGeneratorCtx (AlgId algId=kAlgIdDefault, bool initialize=true) noexcept=0;`	
Parameters (in):	algId	identifier of target RNG algorithm. If no algId is given, the default RNG is returned
	initialize	indicates whether the returned context shall be initialized (i.e., seeded) by the stack
Return value:	ara::core::Result< RandomGeneratorCtx::Uptr >	unique smart pointer to the created RNG context

▽

△

Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if algId argument has an unsupported value or if (alg Id == kAlgIdDefault) and the CryptoProvider does not provide any RandomGeneratorCtx
	SecurityErrorDomain::kBusyResource	if (initialize == true) but the context currently cannot be seeded (e.g., due to a lack of entropy)
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a Random Number Generator (RNG) context.	

⌋*(RS_CRYPTO_02206)*

[SWS_CRYPT_20754]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateSigEncodePrivateCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SigEncodePrivateCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery
	SecurityErrorDomain::kUnknown Identifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a signature encoding private key context.	

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204, RS_AP_00144)*

[SWS_CRYPT_20760]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateSignature(AlgId signAlgId, ReadOnlyMemRegion value, const RestrictedUseObject &key, AlgId hashAlgId=kAlgIdNone)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> CreateSignature (AlgId signAlgId, ReadOnlyMemRegion value, const RestrictedUseObject &key, AlgId hashAlgId=kAlgIdNone) noexcept=0;`	
	signAlgId	identifier of an applied signature/MAC/AE/AEAD crypto algorithm
	value	raw BLOB value of the signature/MAC
	key	symmetric or asymmetric key (according to signAlg Id) applied for the sign or MAC/AE/AEAD operation

▽

△

	hashAlgId	identifier of a hash function algorithm applied together with the signature algorithm
Return value:	ara::core::Result< Signature::Uptrc >	unique smart pointer to the created Signature object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if signAlgId or hashAlgId arguments have unsupported values
	SecurityErrorDomain::kInvalid Argument	if signAlgId or hashAlgId arguments specify crypto algorithms different from the signature/MAC/AE/AEAD and message digest respectively
	SecurityErrorDomain::kIncompatible Arguments	if signAlgId and hashAlgId arguments specify incompatible algorithms (if signAlgId includes hash function specification) or if a crypto primitive associated with the key argument is incompatible with provided signAlgId or hashAlgId arguments
	SecurityErrorDomain::kInvalidInputSize	if the value argument has invalid size (i.e. incompatible with the signAlgId argument)
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Construct Signature object from directly provided components of a digital signature/MAC or authenticated encryption (AE/AEAD). All integers inside a digital signature BLOB value are always presented in Big Endian bytes order (i.e. MSF - Most Significant byte First).	

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006, RS_AP_00144)*

[SWS_CRYPT_20756]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateSignerPrivateCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SignerPrivateCtx::Uptr> CreateSignerPrivateCtx (AlgId algId) noexcept=0;`	
Parameters (in):	algId	identifier of the target signature crypto algorithm
Return value:	ara::core::Result< SignerPrivateCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if algId argument specifies a crypto algorithm different from private key signature
	SecurityErrorDomain::kUnknown Identifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a signature private key context.	

⌋*(RS_CRYPTO_02204, RS_AP_00144)*

[SWS_CRYPT_20744]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_20742]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateSymmetricBlockCipherCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SymmetricBlockCipherCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknownIdentifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric block cipher context.	

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_20743]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateSymmetricKeyWrapperCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SymmetricKeyWrapperCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if algId argument specifies a crypto algorithm different from symmetric key-wrapping
	SecurityErrorDomain::kUnknownIdentifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric key-wrap algorithm context.	

⌋(*RS_CRYPTO_02104, RS_CRYPTO_02208*)

[SWS_CRYPT_20757]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateVerifierPublicCtx(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<VerifierPublicCtx::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if algId argument specifies a crypto algorithm different from public key signature verification
	SecurityErrorDomain::kUnknownIdentifier	if algId argument has an unsupported value
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a signature verification public key context.	

⌋(*RS_CRYPTO_02204, RS_AP_00144*)

[SWS_CRYPT_20710]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02107*)

[SWS_CRYPT_20731]{DRAFT} ⌈

Kind:	function	
Symbol:	ExportPublicObject(const IOInterface &container, Serializable::FormatId format Id=Serializable::kFormatDefault)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > Export PublicObject (const IOInterface &container, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0;`	
Parameters (in):	container	the IOInterface that contains an object for export
	formatId	the CryptoProvider specific identifier of the output format
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	actual capacity required for the serialized data
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kEmpty Container	if the container is empty
	SecurityErrorDomain::kUnexpected Value	if the container contains a secret crypto object
	SecurityErrorDomain::kInsufficient Capacity	if (serialized.empty() == false), but its capacity is not enough for storing result
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Export publicly an object from a IOInterface (i.e. without an intermediate creation of a crypto object).	

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

[SWS_CRYPT_20728]{DRAFT} ⌈

Kind:	function	
Symbol:	ExportSecuredObject(const CryptoObject &object, SymmetricKeyWrapperCtx &transport Context)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > Export SecuredObject (const CryptoObject &object, SymmetricKeyWrapperCtx &transportContext) noexcept=0;`	
Parameters (in):	object	the crypto object for export
	transportContext	the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyExporting)
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	the wrapped crypto object data
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
	SecurityErrorDomain::kIncompatible Object	if the object cannot be exported due to Is Exportable() returning flase
	SecurityErrorDomain::kIncompleteArg State	if the transportContext is not initialized

▽

△

	SecurityErrorDomain::kIncompatible Object	if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Export a crypto object in a secure manner. if (serialized.empty() == true) then the method returns required size only, but content of the transportContext stays unchanged! Only an exportable and completed object (i.e. that have a UUID) can be exported!	

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

[SWS_CRYPT_20729]{DRAFT} ⌈

Kind:	function	
Symbol:	ExportSecuredObject(const IOInterface &container, SymmetricKeyWrapperCtx &transport Context)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > Export SecuredObject (const IOInterface &container, SymmetricKeyWrapperCtx &transportContext) noexcept=0;`	
Parameters (in):	container	the IOInterface that refers an object for export
	transportContext	the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyExporting)
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	actual capacity required for the serialized data
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kEmpty Container	if the container is empty
	SecurityErrorDomain::kInsufficient Capacity	if size of the serialized buffer is not enough for saving the output data
	SecurityErrorDomain::kIncompleteArg State	if the transportContext is not initialized
	SecurityErrorDomain::kIncompatible Object	if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Export securely an object directly from an IOInterface (i.e. without an intermediate creation of a crypto object). if (serialized == nullptr) then the method returns required size only, but content of the transportContext stays unchanged. This method can be used for re-exporting of just imported object but on another transport key.	

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

[SWS_CRYPT_20722]{DRAFT} ⌈

Kind:	function	
Symbol:	GeneratePrivateKey(AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=false, bool isExportable=false)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<PrivateKey::Uptrc> GeneratePrivateKey (AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=false, bool isExportable=false) noexcept=0;`	
Parameters (in):	algId	the identifier of target public-private key crypto algorithm
	allowedUsage	the flags that define a list of allowed transformations' types in which the target key can be used (see constants in scope of RestrictedUseObject)
	isSession	the "session" (or "temporary") attribute for the target key (if true)
	isExportable	the exportability attribute of the target key (if true)
Return value:	ara::core::Result< PrivateKey::Uptrc >	smart unique pointer to the created private key object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if algId has an unsupported value
	SecurityErrorDomain::kIncompatible Arguments	if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Allocate a new private key context of correspondent type and generates the key value randomly. A common COUID should be shared for both private and public keys. Any serializable (i.e. savable/non-session or exportable) key must generate own COUID!	

⌋*(RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02115)*

[SWS_CRYPT_20723]{DRAFT} ⌈

Kind:	function	
Symbol:	GenerateSeed(AlgId algId, SecretSeed::Usage allowedUsage, bool isSession=true, bool isExportable=false)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> GenerateSeed (AlgId algId, SecretSeed::Usage allowedUsage, bool isSession=true, bool isExportable=false) noexcept=0;`	
Parameters (in):	algId	the identifier of target crypto algorithm
	allowedUsage	the lags that define a list of allowed transformations' types in which the target seed can be used (see constants in scope of RestrictedUseObject)
	isSession	the "session" (or "temporary") attribute of the target seed (if true)
	isExportable	the exportability attribute of the target seed (if true)
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to generated SecretSeed object

▽

△

Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if algId has an unsupported value
	SecurityErrorDomain::kIncompatible Arguments	if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Generate a random Secret Seed object of requested algorithm.	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_20721]{DRAFT} ⌈

Kind:	function	
Symbol:	GenerateSymmetricKey(AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=true, bool isExportable=false)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> GenerateSymmetricKey (AlgId algId, AllowedUsageFlags allowedUsage, bool isSession=true, bool isExportable=false) noexcept=0;`	
Parameters (in):	algId	the identifier of target symmetric crypto algorithm
	allowedUsage	the flags that define a list of allowed transformations' types in which the target key can be used (see constants in scope of RestrictedUseObject)
	isSession	the "session" (or "temporary") attribute of the target key (if true)
	isExportable	the exportability attribute of the target key (if true)
Return value:	ara::core::Result< Symmetric Key::Uptrc >	smart unique pointer to the created symmetric key object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if algId has an unsupported value
	SecurityErrorDomain::kIncompatible Arguments	if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Allocate a new symmetric key object and fill it by a new randomly generated value. Any serializable (i.e. savable/non-session or exportable) key must generate own COUID! By default Crypto Provider should use an internal instance of a best from all supported RNG (ideally TRNG).	

⌋*(RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02115)*

[SWS_CRYPT_20725]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPayloadStorageSize(CryptoObjectType cryptoObjectType, AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	virtual ara::core::Result<std::size_t> GetPayloadStorageSize (CryptoObjectType cryptoObjectType, AlgId algId) const noexcept=0;	
Parameters (in):	algId	a CryptoProvider algorithm ID of the target object
DIRECTION NOT DEFINED	cryptoObjectType	–
Return value:	ara::core::Result< std::size_t >	minimal size required for storing of the object in a TrustedContainer (persistent or volatile)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknownIdentifier	if any argument has an unsupported value
	SecurityErrorDomain::kIncompatibleArguments	if the arguments are incompatible
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Return minimally required capacity of a key slot for saving of the object's payload. Returned value does not take into account the object's meta-information properties, but their size is fixed and common for all crypto objects independently from their actual type. During an allocation of a TrustedContainer, Crypto Providers (and Key Storage Providers) reserve space for an object's meta-information automatically, according to their implementation details.	

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

[SWS_CRYPT_20724]{DRAFT} ⌈

Kind:	function	
Symbol:	GetSerializedSize(CryptoObjectType cryptoObjectType, AlgId algId, Serializable::FormatId formatId=Serializable::kFormatDefault)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	virtual ara::core::Result<std::size_t> GetSerializedSize (CryptoObjectType cryptoObjectType, AlgId algId, Serializable::FormatId formatId=Serializable::kFormatDefault) const noexcept=0;	
Parameters (in):	algId	the Crypto Provider algorithm ID of the target object
	formatId	the Crypto Provider specific identifier of the output format
DIRECTION NOT DEFINED	cryptoObjectType	–
Return value:	ara::core::Result< std::size_t >	size required for storing of the object serialized in the specified format
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknownIdentifier	if any argument has an unsupported value
	SecurityErrorDomain::kIncompatibleArguments	if any pair of the arguments are incompatible
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Return required buffer size for serialization of an object in specific format.	

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

[SWS_CRYPT_20732]{DRAFT} ⌈

Kind:	function	
Symbol:	ImportPublicObject(IOInterface &container, ReadOnlyMemRegion serialized, CryptoObjectType expectedObject=CryptoObjectType::kUndefined)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ImportPublicObject (IOInterface &container, ReadOnlyMemRegion serialized, CryptoObjectType expectedObject=CryptoObjectType::kUndefined) noexcept=0;`	
Parameters (in):	serialized	the memory region that contains a securely serialized object that should be imported to the IOInterface
	expectedObject	the expected object type (default value CryptoObjectType::kUnknown means without check)
Parameters (out):	container	the IOInterface for storing of the imported object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnexpectedValue	if the serialized contains incorrect data
	SecurityErrorDomain::kBadCryptoObjectType	if (expectedObject != CryptoObjectType::kUnknown), but the actual object type differs from the expected one
	SecurityErrorDomain::kInsufficientCapacity	if capacity of the container is not enough to save the de-serialized object
	SecurityErrorDomain::kModifiedResource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	SecurityErrorDomain::kUnreservedResource	if the IOInterface is not opened writable.
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Import publicly serialized object to a storage location pointed to by an IOInterface for following processing (without allocation of a crypto object). If (expectedObject != CryptoObjectType::kUnknown) and an actual object type differs from the expected one then this method fails. If the serialized contains incorrect data then this method fails.	

⌋(*RS_CRYPTO_02105*, *RS_CRYPTO_02112*)

[SWS_CRYPT_20730]{DRAFT} ⌈

Kind:	function	
Symbol:	ImportSecuredObject(IOInterface &container, ReadOnlyMemRegion serialized, SymmetricKeyWrapperCtx &transportContext, bool isExportable=false, CryptoObjectType expectedObject=CryptoObjectType::kUndefined)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ImportSecuredObject (IOInterface &container, ReadOnlyMemRegion serialized, SymmetricKeyWrapperCtx &transportContext, bool isExportable=false, CryptoObjectType expectedObject=CryptoObjectType::kUndefined) noexcept=0;`	
	serialized	the memory region that contains a securely serialized object that should be imported to the IOInterface
	transportContext	the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyImporting)

▽

△

	isExportable	the exportability attribute of the target object
	expectedObject	the expected object type (default value CryptoObject Type::kUnknown means without check)
Parameters (out):	container	the IOInterface for storing of the imported object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnexpected Value	if the serialized contains incorrect data
	SecurityErrorDomain::kBadCrypto ObjectType	if (expectedObject != CryptoObjectType::k Unknown), but the actual object type differs from the expected one
	SecurityErrorDomain::kIncompleteArg State	if the transportContext is not initialized
	SecurityErrorDomain::kIncompatible Object	if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)
	SecurityErrorDomain::kInsufficient Capacity	if capacity of the container is not enough to save the deserialized object
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	SecurityErrorDomain::kUnreserved Resource	if the IOInterface is not opened writeable.
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Import securely serialized object to the persistent or volatile storage represented by an IOInterface for following processing.	

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

[SWS_CRYPT_20733]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadObject(const IOInterface &container)	
Scope:	class ara::crypto::cryp::CryptoProvider	
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< CryptoObject::Uptrc >	unique smart pointer to the created object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
	SecurityErrorDomain::kEmpty Container	if the container is empty
	SecurityErrorDomain::kResourceFault	if the container content is damaged
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.

▽

△

	SecurityErrorDomain::kIncompatible Object	if the underlying resource belongs to another, incompatible CryptoProvider
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load any crypto object from the IOInterface provided.	
Notes:	This method is one of the "binding" methods between a CryptoProvider and the Key Storage Provider.	

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02002*, *RS_CRYPTO_02005*, *RS_CRYPTO_02006*)

[SWS_CRYPT_20764]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadPrivateKey(const IOInterface &container)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<PrivateKey::Uptrc> LoadPrivateKey (const IOInterface &container) noexcept=0;`	
Parameters (in):	container	the IOInterface that contains the crypto object for loading
Return value:	ara::core::Result< PrivateKey::Uptrc >	unique smart pointer to the PrivateKey
Exception Safety:	noexcept	
Errors:	SecurityErrorDomain::kEmpty Container	if the container is empty
	SecurityErrorDomain::kResourceFault	if the container content is damaged
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	SecurityErrorDomain::kIncompatible Object	if the underlying resource belongs to another, incompatible CryptoProvider
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load a private key from the IOInterface provided.	

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02002*)

[SWS_CRYPT_20763]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadPublicKey(const IOInterface &container)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<PublicKey::Uptrc> LoadPublicKey (const IOInterface &container) noexcept=0;`	
Parameters (in):	container	the IOInterface that contains the crypto object for loading
Return value:	ara::core::Result< PublicKey::Uptrc >	unique smart pointer to the PublicKey
Exception Safety:	noexcept	
Errors:	SecurityErrorDomain::kEmpty Container	if the container is empty

▽

△

	SecurityErrorDomain::kResourceFault	if the container content is damaged
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	SecurityErrorDomain::kIncompatible Object	if the underlying resource belongs to another, incompatible CryptoProvider
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load a public key from the IOInterface provided.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002)*

[SWS_CRYPT_20765]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadSecretSeed(const IOInterface &container)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> LoadSecretSeed (const IOInterface &container) noexcept=0;`	
Parameters (in):	container	the IOInterface that contains the crypto object for loading
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to the SecretSeed
Exception Safety:	noexcept	
Errors:	SecurityErrorDomain::kEmpty Container	if the container is empty
	SecurityErrorDomain::kResourceFault	if the container content is damaged
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	SecurityErrorDomain::kIncompatible Object	if the underlying resource belongs to another, incompatible CryptoProvider
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load secret seed from the IOInterface provided.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002)*

[SWS_CRYPT_20762]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadSymmetricKey(const IOInterface &container)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> LoadSymmetricKey (const IOInterface &container) noexcept=0;`	
Parameters (in):	container	the IOInterface that contains the crypto object for loading
Return value:	ara::core::Result< Symmetric Key::Uptrc >	unique smart pointer to the SymmetricKey
Exception Safety:	noexcept	

▽

△

Errors:	SecurityErrorDomain::kEmpty Container	if the container is empty
	SecurityErrorDomain::kResourceFault	if the container content is damaged
	SecurityErrorDomain::kModified Resource	if the underlying resource has been modified after the IOInterface has been opened, i.e., the IOInterface has been invalidated.
	SecurityErrorDomain::kIncompatible Object	if the underlying resource belongs to another, incompatible CryptoProvider
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load a symmetric key from the IOInterface provided.	

⌋(*RS_CRYPTO_02001, RS_CRYPTO_02002*)

[SWS_CRYPT_29023]{DRAFT} ⌈

Kind:	function	
Symbol:	GetBlockSize()	
Scope:	class ara::crypto::cryp::CryptoService	
Syntax:	`virtual std::size_t GetBlockSize () const noexcept=0;`	
Return value:	std::size_t	size of the block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_service.h"	
Description:	Get block (or internal buffer) size of the base algorithm. For digest, byte-wise stream cipher and RNG contexts it is an informative method, intended only for optimization of the interface usage.	

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_29021]{DRAFT} ⌈

Kind:	function	
Symbol:	GetMaxInputSize(bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::CryptoService	
Syntax:	`virtual std::size_t GetMaxInputSize (bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	suppressPadding	if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only
Return value:	std::size_t	maximum size of the input data block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_service.h"	
Description:	Get maximum expected size of the input data block. suppressPadding argument and it will be equal to the block size.	

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_29022]{DRAFT} ⌈

Kind:	function	
Symbol:	GetMaxOutputSize(bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::CryptoService	
Syntax:	`virtual std::size_t GetMaxOutputSize (bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	suppressPadding	if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only
Return value:	std::size_t	maximum size of the output data block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_service.h"	
Description:	Get maximum possible size of the output data block. If (IsEncryption() == true) then a value returned by this method is independent from the suppressPadding argument and will be equal to the block size.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_30216]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const CryptoProvider &other)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`CryptoProvider& operator= (const CryptoProvider &other)=default;`	
Parameters (in):	other	the other instance
Return value:	CryptoProvider &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Copy-assign another CryptoProvider to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30217]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(CryptoProvider &&other)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`CryptoProvider& operator= (CryptoProvider &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	CryptoProvider &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Move-assign another CryptoProvider to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_20802]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCryptoService()	
Scope:	class ara::crypto::cryp::DecryptorPrivateCtx	
Syntax:	`virtual CryptoService::Uptr GetCryptoService () const noexcept=0;`	
Return value:	CryptoService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"	
Description:	Get CryptoService instance.	

⌋(*RS_CRYPTO_02006*)

[SWS_CRYPT_20812]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::DecryptorPrivateCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ProcessBlock (ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	in	the input data block
	suppressPadding	if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInputSize	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kInsufficientCapacity	if the out.size() is not enough to store the transformation result
	SecurityErrorDomain::kUninitializedContext	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutputSize(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_20813]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::DecryptorPrivateCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ProcessBlock (ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	the input data block
	suppressPadding	if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data
Return value:	ara::core::Result< ByteVector< Alloc > >	the managed container for output block
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInput Size	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kInsufficient Capacity	if the out.size() is not enough to store the transformation result
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_20811]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::DecryptorPrivateCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"	
Description:	Clear the crypto context.	

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_20810]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_29013]{DRAFT} ⌈

Kind:	function	
Symbol:	Compare(ReadOnlyMemRegion expected, std::size_t offset=0)	
Scope:	class ara::crypto::cryp::DigestService	
Syntax:	`virtual ara::core::Result<bool> Compare (ReadOnlyMemRegion expected, std::size_t offset=0) const noexcept=0;`	
Parameters (in):	expected	the memory region containing an expected digest value
	offset	position of the first byte in calculated digest for the comparison starting
Return value:	ara::core::Result< bool >	true if the expected bytes sequence is identical to first bytes of calculated digest
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Finished	if the digest calculation was not finished by a call of the Finish() method
	SecurityErrorDomain::kBruteForceRisk	if the buffered digest belongs to a MAC/HMAC/AE/AEAD context, which was initialized by a key without kAllowSignature permission, but actual size of requested digest is less than 8 bytes (it is a protection from the brute-force attack)
Header file:	#include "ara/crypto/cryp/digest_service.h"	
Description:	Compare the calculated digest against an expected value. Entire digest value is kept in the context up to next call Start(), therefore any its part can be verified again or extracted. If (full_digest_size <= offset) || (expected.size() == 0) then return false; else comparison_size = min(expected.size(), (full_digest_size - offset)) bytes. This method can be implemented as "inline" after standartization of function ara::core::memcmp().	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29012]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_29015]{DRAFT} ⌈

Kind:	function	
Symbol:	IsFinished()	
Scope:	class ara::crypto::cryp::DigestService	
Syntax:	`virtual bool IsFinished () const noexcept=0;`	
Return value:	bool	true if a previously started stream processing was finished by a call of the Finish() or FinishBytes() methods
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/digest_service.h"	
Description:	Check current status of the stream processing: finished or no.	

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_29014]{DRAFT} ⌈

Kind:	function	
Symbol:	IsStarted()	
Scope:	class ara::crypto::cryp::DigestService	
Syntax:	`virtual bool IsStarted () const noexcept=0;`	
Return value:	bool	true if the processing was start by a call of the Start() methods and was not finished yet
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/digest_service.h"	
Description:	Check current status of the stream processing: started or no.	

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_21002]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCryptoService()	
Scope:	class ara::crypto::cryp::EncryptorPublicCtx	
Syntax:	`virtual CryptoService::Uptr GetCryptoService () const noexcept=0;`	
Return value:	CryptoService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Description:	Get CryptoService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_21012]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::EncryptorPublicCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ProcessBlock (ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	in	the input data block
	suppressPadding	if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInputSize	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kUninitializedContext	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutputSize(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_21013]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::EncryptorPublicCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ProcessBlock (ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	the input data block
	suppressPadding	if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data
Return value:	ara::core::Result< ByteVector< Alloc > >	the managed container for output block
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInput Size	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kInsufficient Capacity	if the out.size() is not enough to store the transformation result
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_21011]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::EncryptorPublicCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_21010]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_29041]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29045]{DRAFT} ⌈

Kind:	function	
Symbol:	GetActualKeyBitLength()	
Scope:	class ara::crypto::cryp::ExtensionService	
Syntax:	`virtual std::size_t GetActualKeyBitLength () const noexcept=0;`	
Return value:	std::size_t	actual length of a key (now set to the algorithm context) in bits
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Description:	Get actual bit-length of a key loaded to the context. If no key was set to the context yet then 0 is returned.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29047]{DRAFT} ⌈

Kind:	function	
Symbol:	GetActualKeyCOUID()	
Scope:	class ara::crypto::cryp::ExtensionService	
Syntax:	`virtual CryptoObjectUid GetActualKeyCOUID () const noexcept=0;`	
Return value:	CryptoObjectUid	the COUID of the CryptoObject
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Description:	Get the COUID of the key deployed to the context this extension service is attached to. If no key was set to the context yet then an empty COUID (Nil) is returned.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29046]{DRAFT} ⌈

Kind:	function	
Symbol:	GetAllowedUsage()	
Scope:	class ara::crypto::cryp::ExtensionService	
Syntax:	`virtual AllowedUsageFlags GetAllowedUsage () const noexcept=0;`	
Return value:	AllowedUsageFlags	a combination of bit-flags that specifies allowed usages of the context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Description:	Get allowed usages of this context (according to the key object attributes loaded to this context). If the context is not initialized by a key object yet then zero (all flags are reset) must be returned.	

⌋*(RS_CRYPTO_02008)*

[SWS_CRYPT_29044]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29043]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29048]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29049]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_30218]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const ExtensionService &other)	
Scope:	class ara::crypto::cryp::ExtensionService	
Syntax:	`ExtensionService& operator= (const ExtensionService &other)=default;`	
Parameters (in):	other	the other instance
Return value:	ExtensionService &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Description:	Copy-assign another ExtensionService to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30219]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(ExtensionService &&other)	
Scope:	class ara::crypto::cryp::ExtensionService	
Syntax:	`ExtensionService& operator= (ExtensionService &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	ExtensionService &	*this, containing the contents of other
Header file:	#include "ara/crypto/cryp/extension_service.h"	
Description:	Move-assign another ExtensionService to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_21115]{DRAFT} ⌈

Kind:	function	
Symbol:	Finish()	
Scope:	class ara::crypto::cryp::HashFunctionCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > Finish () noexcept=0;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	unique smart pointer to created signature object, if (makeSignatureObject == true) or an empty Signature object if (makeSignatureObject == false)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Started	if the digest calculation was not initiated by a call of the Start() method
	SecurityErrorDomain::kInvalidUsage Order	if the digest calculation has not started yet or not been updated at least once
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Description:	Finish the digest calculation and optionally produce the "signature" object. Only after call of this method the digest can be signed, verified, extracted or compared.	

⌋(*RS_CRYPTO_02302*, *RS_CRYPTO_02205*)

[SWS_CRYPT_21102]{DRAFT} ⌈

Kind:	function	
Symbol:	GetDigestService()	
Scope:	class ara::crypto::cryp::HashFunctionCtx	
Syntax:	`virtual DigestService::Uptr GetDigestService () const noexcept=0;`	
Return value:	DigestService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Description:	Get DigestService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_21116]{DRAFT} ⌈

Kind:	function	
Symbol:	GetDigest(std::size_t offset=0)	
Scope:	class ara::crypto::cryp::HashFunctionCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > GetDigest (std::size_t offset=0) const noexcept=0;`	
Parameters (in):	offset	position of the first byte of digest that should be placed to the output buffer
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	number of digest bytes really stored to the output buffer (they are always <= output.size() and denoted below as return_size)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNotFinished	if the digest calculation was not finished by a call of the Finish() method
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Description:	Get requested part of calculated digest. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.size(), (full_digest_size - offset)) bytes. This method can be implemented as "inline" after standartization of function ara::core::memcpy().	

⌋*(RS_CRYPTO_02205)*

[SWS_CRYPT_21117]{DRAFT} ⌈

Kind:	function	
Symbol:	GetDigest(std::size_t offset=0)	
Scope:	class ara::crypto::cryp::HashFunctionCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > GetDigest (std::size_t offset=0) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	offset	position of first byte of digest that should be placed to the output buffer

▽

△

Return value:	ara::core::Result< ByteVector< Alloc > >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Finished	if the digest calculation was not finished by a call of the Finish() method
	SecurityErrorDomain::kUsageViolation	if the buffered digest belongs to a MAC/HMAC/AE/ AEAD context initialized by a key without kAllow Signature permission
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Description:	Get requested part of calculated digest to pre-reserved managed container. This method sets the size of the output container according to actually saved value. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.capacity(), (full_digest_size - offset)) bytes.	

⌋*(RS_CRYPTO_02205)*

[SWS_CRYPT_21118]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_21110]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(ReadOnlyMemRegion iv)	
Scope:	class ara::crypto::cryp::HashFunctionCtx	
Syntax:	`virtual ara::core::Result<void> Start (ReadOnlyMemRegion iv) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)

▽

△

	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false)
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Description:	Initialize the context for a new data stream processing or generation (depending on the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_21111]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(const SecretSeed &iv)	
Scope:	class ara::crypto::cryp::HashFunctionCtx	
Syntax:	`virtual ara::core::Result<void> Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Description:	Initialize the context for a new data stream processing or generation (depending on the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_21112]{DRAFT} ⌈

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

▽

△

Description:	Update the digest calculation context by a new part of the message. This method is dedicated for cases then the RestrictedUseObject is a part of the "message".

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_21113]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_21114]{DRAFT} ⌈

Kind:	function	
Symbol:	Update(std::uint8_t in)	
Scope:	class ara::crypto::cryp::HashFunctionCtx	
Syntax:	`virtual ara::core::Result<void> Update (std::uint8_t in) noexcept=0;`	
Parameters (in):	in	a byte value that is a part of input message
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Started	if the digest calculation was not initiated by a call of the Start() method
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"	
Description:	Update the digest calculation context by a new part of the message. This method is convenient for processing of constant tags.	

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_21312]{DRAFT} ⌈

Kind:	function	
Symbol:	AgreeKey(const PublicKey &otherSideKey, KeyDerivationFunctionCtx &kdf, AlgId targetAlgId, AllowedUsageFlags allowedUsage, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> AgreeKey (const PublicKey &otherSideKey, KeyDerivationFunctionCtx &kdf, AlgId targetAlgId, AllowedUsageFlags allowedUsage, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	otherSideKey	the public key of the other side of the Key-Agreement
	kdf	the Context of a Key Derivation Function, which should be used for the target key production
	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 SecretSeed object, which contains the key material produced by the Key-Agreement algorithm
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitializedContext	if the context was not initialized by a key value
	SecurityErrorDomain::kIncompatibleObject	if the public and private keys correspond to different algorithms
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Description:	Produce a common symmetric key via execution of the key-agreement algorithm between this private key and a public key of another side. Produced SymmetricKey object has following attributes: session, non-exportable. This method can be used for direct production of the target key, without creation of the intermediate SecretSeed object.	

⌋(*RS_CRYPTO_02115*)

[SWS_CRYPT_21311]{DRAFT} ⌈

Kind:	function	
Symbol:	AgreeSeed(const PublicKey &otherSideKey, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage)	
Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> AgreeSeed (const PublicKey &otherSideKey, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage) const noexcept=0;`	
Parameters (in):	otherSideKey	the public key of the other side of the Key-Agreement
	allowedUsage	the allowed usage scope of the target seed

▽

△

Return value:	ara::core::Result< SecretSeed::Uptrc >	unique pointer to SecretSeed object, which contains the key material produced by the Key-Agreement algorithm
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
	SecurityErrorDomain::kIncompatible Object	if the public and private keys correspond to different algorithms
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Description:	Produce a common secret seed via execution of the key-agreement algorithm between this private key and a public key of another side. Produced SecretSeed object has following attributes: session, non-exportable, AlgID (this Key-Agreement Algorithm ID).	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_21302]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Description:	Get ExtensionService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_21314]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_21313]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_21412]{DRAFT} ⌈

Kind:	function	
Symbol:	DecapsulateKey(ReadOnlyMemRegion input, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=Read OnlyMemRegion())	
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> DecapsulateKey (Read OnlyMemRegion input, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctx Label=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	input	an input buffer (its size should be equal Get EncapsulatedSize() bytes)
	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
	salt	an optional salt value (if used, it should be unique for each instance of the target key)
	ctxLabel	an pptional application specific "context label" (it can identify purpose of the target key and/or communication parties)
Return value:	ara::core::Result< Symmetric Key::Uptrc >	unique smart pointer to a symmetric key object derived from a key material decapsulated from the input block
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a private key value
	SecurityErrorDomain::kUnknown Identifier	if kekAlgId specifies incorrect algorithm

▽

△

	SecurityErrorDomain::kInvalidInputSize	if (input.size() <> this->GetEncapsulatedSize())
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Description:	Decapsulate Key Encryption Key (KEK). Produced SymmetricKey object has following attributes: session, non-exportable, Key Usage: kAllowKeyImporting. This method can be used for direct production of the target key, without creation of the intermediate SecretSeed object.	

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115)*

[SWS_CRYPT_21411]{DRAFT} ⌈

Kind:	function	
Symbol:	DecapsulateSeed(ReadOnlyMemRegion input, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage)	
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> DecapsulateSeed (ReadOnlyMemRegion input, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage) 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
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to SecretSeed object, which keeps the key material decapsulated from the input buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitializedContext	if the context was not initialized by a private key value
	SecurityErrorDomain::kInsufficientCapacity	if the output.size() is not enough to save the decapsulation result
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Description:	Decapsulate key material. Returned Key Material object should be used for derivation of a symmetric key. Produced SecretSeed object has following attributes: session, non-exportable, AlgID = this KEM AlgID.	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_21416]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_21402]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Description:	Get ExtensionService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_21415]{DRAFT} ⌈

Kind:	function	
Symbol:	GetKekEntropy()	
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx	
Syntax:	`virtual std::size_t GetKekEntropy () const noexcept=0;`	
Return value:	std::size_t	entropy of the KEK material in bits
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Description:	Get entropy (bit-length) of the key encryption key (KEK) material. For RSA system the returned value corresponds to the length of module N (minus 1). For DH-like system the returned value corresponds to the length of module q (minus 1).	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_21414]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_21413]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const PrivateKey &key)	
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this private key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Description:	Set (deploy) a key to the key decapsulator private algorithm context.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_21510]{DRAFT} ⌈

Kind:	function	
Symbol:	AddSalt(ReadOnlyMemRegion salt)	
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 >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if size of the appFiller is incorrect, i.e. if (app Filler.size() < GetFillerSize());
	SecurityErrorDomain::kInvalidInputSize	if size of the appFiller is incorrect, i.e. if (app Filler.size() < GetFillerSize());
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Add an application filler value stored in a (non-secret) ReadOnlyMemRegion. If (GetFillerSize() == 0), then this method call will be ignored. Add a secret application filler value stored in a SecretSeed object. If (GetFillerSize() == 0), then this method call will be ignored. Add a salt value stored in a (non-secret) ReadOnlyMemRegion.	

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111)*

[SWS_CRYPT_21513]{DRAFT} ⌈

Kind:	function	
Symbol:	AddSecretSalt(const SecretSeed &salt)	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<void> AddSecretSalt (const SecretSeed &salt) noexcept=0;`	
Parameters (in):	salt	a salt value (if used, it should be unique for each instance of the target key)
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Add a secret salt value stored in a SecretSeed object.	

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_-02111)*

[SWS_CRYPT_21514]{DRAFT} ⌈

Kind:	function	
Symbol:	ConfigIterations(std::uint32_t iterations=0)	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual std::uint32_t ConfigIterations (std::uint32_t iterations=0) noexcept=0;`	
Parameters (in):	iterations	the required number of iterations of the base function (0 means implementation default number)
Return value:	std::uint32_t	actual number of the iterations configured in the context now (after this method call)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Configure the number of iterations that will be applied by default. Implementation can restrict minimal and/or maximal value of the iterations number.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_21515]{DRAFT} ⌈

Kind:	function	
Symbol:	DeriveKey(bool isSession=true, bool isExportable=false)	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> DeriveKey (bool isSession=true, bool isExportable=false) const noexcept=0;`	
Parameters (in):	isSession	the "session" (or "temporary") attribute for the target key (if true)
	isExportable	the exportability attribute for the target key (if true)
Return value:	ara::core::Result< SymmetricKey::Uptrc >	unique smart pointer to the created instance of derived symmetric key

▽

△

Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not sufficiently initialized
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Derive a symmetric key from the provided key material and provided context configuration.	

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02115)*

[SWS_CRYPT_21516]{DRAFT} ⌈

Kind:	function	
Symbol:	DeriveSeed(bool isSession=true, bool isExportable=false)	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> DeriveSeed (bool isSession=true, bool isExportable=false) const noexcept=0;`	
Parameters (in):	isSession	the "session" (or "temporary") attribute for the target key (if true)
	isExportable	the exportability attribute for the target key (if true)
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to the created SecretSeed object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not sufficiently initialized
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Derive a "slave" key material (secret seed) from the provided "master" key material and provided context configuration.	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_21524]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_21517]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Get ExtensionService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_21518]{DRAFT} ⌈

Kind:	function	
Symbol:	GetKeyIdSize()	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual std::size_t GetKeyIdSize () const noexcept=0;`	
Return value:	std::size_t	size of the application specific filler in bytes Returned value is constant for this instance of the key derivation context, i.e. independent from configuration by the @c Init() call. size of the key ID in bytes the @c Init() call.
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Get the fixed size of an application specific "filler" required by this context instance. If this instance of the key derivation context does not support filler values, 0 shall be returned. Get the fixed size of the target key ID required by diversification algorithm. Returned value is constant for each instance of the interface, i.e. independent from configuration by	

⌋*(RS_CRYPTO_02103, RS_CRYPTO_02103)*

[SWS_CRYPT_21520]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02103)*

[SWS_CRYPT_21521]{DRAFT} ⌈

Kind:	function	
Symbol:	GetTargetAllowedUsage()	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual AllowedUsageFlags GetTargetAllowedUsage () const noexcept=0;`	
Return value:	AllowedUsageFlags	allowed key usage bit-flags of target keys
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Get allowed key usage of target (slave) key. The returned value depends on the source key-material allowed usage flags and the argument allowedUsage of last call of the Init() method. If the context has not yet been configured by a call of the Init() method, the allowed usage flags of the source key-material shall be returned. If the context has not yet been configured by a call of the Init() method and no source key-material has been set either, kAllowKdfMaterialAnyUsage shall be returned.	

⌋*(RS_CRYPTO_02008)*

[SWS_CRYPT_21522]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02103)*

[SWS_CRYPT_21523]{DRAFT} ⌈

Kind:	function	
Symbol:	Init(ReadOnlyMemRegion targetKeyId, AlgId targetAlgId=kAlgIdAny, AllowedUsageFlags allowedUsage=kAllowKdfMaterialAnyUsage, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<void> Init (ReadOnlyMemRegion targetKeyId, AlgId targetAlgId=kAlgIdAny, AllowedUsageFlags allowedUsage=kAllowKdfMaterialAnyUsage, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion()) noexcept=0;`	
	targetKeyId	ID of the target key
	targetAlgId	the identifier of the target symmetric crypto algorithm

▽

△

	allowedUsage	bit-flags that define a list of allowed transformations' types in which the target key may be used
	ctxLabel	an optional application specific "context label" (this can identify the purpose of the target key and/or communication parties)
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Arguments	if targetAlgId specifies a cryptographic algorithm different from a symmetric one with key length equal to GetTargetKeyBitLength();
	SecurityErrorDomain::kUsageViolation	if allowedUsage specifies more usages of the derived key-material than the source key-material, i.e. usage of the derived key-material may not be expanded beyond what the source key-material allows
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Initialize this context by setting at least the target key ID. The byte sequence provided via argument ctxLabel can include a few fields with different meaning separated by single 0x00 byte. If (targetAlgId == kAlgIdAny) then a diversified key can be loaded to any symmetric context that supports the same key length (if the "allowed usage" flags are also satisfied)!	

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_-02111, RS_CRYPTO_02115)*

[SWS_CRYPT_21525]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_21818]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_21802]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Get ExtensionService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_21817]{DRAFT} ⌈

Kind:	function	
Symbol:	GetKekEntropy()	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`virtual std::size_t GetKekEntropy () const noexcept=0;`	
Return value:	std::size_t	entropy of the KEK material in bits
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Get entropy (bit-length) of the key encryption key (KEK) material. For RSA system the returned value corresponds to the length of module N (minus 1). For DH-like system the returned value corresponds to the length of module q (minus 1).	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_21810]{DRAFT} ⌈

Kind:	function	
Symbol:	AddKeyingData(RestrictedUseObject &keyingData)	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`virtual ara::core::Result<void> AddKeyingData (RestrictedUseObject &keyingData) noexcept=0;`	
Parameters (in):	keyingData	the payload to be protected
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Add the content to be encapsulated (payload) according to RFC 5990 ("keying data"). At the moment only SymmetricKey and SecretSeed objects are supported.	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_21813]{DRAFT} ⌈

Kind:	function	
Symbol:	Encapsulate(KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > Encapsulate (KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion()) 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
	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< ara::core::Vector< ara::core::Byte > >	unique smart pointer to a symmetric key object derived from a randomly generated material encapsulated to the output buffer Only first GetEncapsulatedSize() bytes of the output buffer should be updated by this method. Produced SymmetricKey object has following attributes: session, non-exportable, Allowed Key Usage: kAllowKeyExporting. This method can be used for direct production of the target key, without creation of the intermediate SecretSeed object.
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitializedContext	if the context was not initialized by a public key value
	SecurityErrorDomain::kInvalidArgument	if kekAlgId specifies incorrect algorithm
	SecurityErrorDomain::kInsufficientCapacity	if the output.size() is not enough to save the encapsulation result

▽

△

Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"
Description:	Encapsulate Key Encryption Key (KEK).

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115)*

[SWS_CRYPT_21816]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_21815]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const PublicKey &key)	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const PublicKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Set (deploy) a key to the key encapsulator public algorithm context.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_22119]{DRAFT} ⌈

Kind:	function	
Symbol:	Check(const Signature &expected)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<bool> Check (const Signature &expected) const noexcept=0;`	
Parameters (in):	expected	the signature object containing an expected digest value
Return value:	ara::core::Result< bool >	true if value and meta-information of the provided "signature" object is identical to calculated digest and current configuration of the context respectively; but false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Finished	if the digest calculation was not finished by a call of the Finish() method
	SecurityErrorDomain::kIncompatible Object	if the provided "signature" object was produced by another crypto primitive type
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Check the calculated digest against an expected "signature" object. Entire digest value is kept in the context up to next call Start(), therefore it can be verified again or extracted. This method can be implemented as "inline" after standartization of function ara::core::memcmp().	

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02204)*

[SWS_CRYPT_22115]{DRAFT} ⌈

Kind:	function	
Symbol:	Finish(bool makeSignatureObject=false)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> Finish (bool makeSignatureObject=false) noexcept=0;`	
Parameters (in):	makeSignatureObject	if this argument is true then the method will also produce the signature object
Return value:	ara::core::Result< Signature::Uptrc >	unique smart pointer to created signature object, if (makeSignatureObject == true) or nullptr if (makeSignatureObject == false)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Started	if the digest calculation was not initiated by a call of the Start() method
	SecurityErrorDomain::kUsageViolation	if the buffered digest belongs to a MAC/HMAC/AE/AEAD context initialized by a key without kAllowSignature permission, but (makeSignatureObject == true)
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Finish the digest calculation and optionally produce the "signature" object. Only after call of this method the digest can be signed, verified, extracted or compared! If the signature object produced by a keyed MAC/HMAC/AE/AEAD algorithm then the dependence COUID of the "signature" should be set to COUID of used symmetric key.	

⌋*(RS_CRYPTO_02302, RS_CRYPTO_02203)*

[SWS_CRYPT_22102]{DRAFT} ⌈

Kind:	function	
Symbol:	GetDigestService()	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual DigestService::Uptr GetDigestService () const noexcept=0;`	
Return value:	DigestService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Get DigestService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_22116]{DRAFT} ⌈

Kind:	function	
Symbol:	GetDigest(std::size_t offset=0)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > GetDigest (std::size_t offset=0) const noexcept=0;`	
Parameters (in):	offset	position of the first byte of digest that should be placed to the output buffer
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	number of digest bytes really stored to the output buffer (they are always <= output.size() and denoted below as return_size)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNotFinished	if the digest calculation was not finished by a call of the Finish() method
	SecurityErrorDomain::kUsageViolation	if the buffered digest belongs to a MAC/HMAC/AE/AEAD context initialized by a key without kAllowSignature permission
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Get requested part of calculated digest to existing memory buffer. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.size(), (full_digest_size - offset)) bytes. This method can be implemented as "inline" after standartization of function ara::core::memcpy().	

⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_22117]{DRAFT} ⌈

Kind:	function
Symbol:	GetDigest(std::size_t offset=0)
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > GetDigest (std::size_t offset=0) const noexcept;`

▽

△

Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	offset	position of first byte of digest that should be placed to the output buffer
Return value:	ara::core::Result< ByteVector< Alloc > >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNotFinished	if the digest calculation was not finished by a call of the Finish() method
	SecurityErrorDomain::kUsageViolation	if the buffered digest belongs to a MAC/HMAC/AE/AEAD context initialized by a key without kAllowSignature permission
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Get requested part of calculated digest to pre-reserved managed container. This method sets the size of the output container according to actually saved value. Entire digest value is kept in the context up to next call Start(), therefore any its part can be extracted again or verified. If (full_digest_size <= offset) then return_size = 0 bytes; else return_size = min(output.capacity(), (full_digest_size - offset)) bytes.	

⌋*(RS_CRYPTO_02203)*

[SWS_CRYPT_22120]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_22118]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kMacGenerate)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kMacGenerate) noexcept=0;`	
Parameters (in):	key	the source key object
DIRECTION NOT DEFINED	transform	–
Return value:	ara::core::Result< void >	–

▽

△

Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Set (deploy) a key to the message authn code algorithm context.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_22110]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(ReadOnlyMemRegion iv=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<void> Start (ReadOnlyMemRegion iv=ReadOnlyMemRegion()) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by deploying a key
	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false)
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_22111]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(const SecretSeed &iv)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<void> Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object

▽

△

Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by deploying a key
	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation
	SecurityErrorDomain::kUsageViolation	if this transformation type is prohibited by the "allowed usage" restrictions of the provided Secret Seed object
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_22112]{DRAFT} ⌈

Kind:	function	
Symbol:	Update(const RestrictedUseObject &in)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<void> Update (const RestrictedUseObject &in) noexcept=0;`	
Parameters (in):	in	a part of input message that should be processed
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Started	if the digest calculation was not initiated by a call of the Start() method
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Update the digest calculation context by a new part of the message. This method is dedicated for cases then the RestrictedUseObject is a part of the "message".	

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_22113]{DRAFT} ⌈

Kind:	function	
Symbol:	Update(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<void> Update (ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	a part of the input message that should be processed

▽

△

Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Started	if the digest calculation was not initiated by a call of the Start() method
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Update the digest calculation context by a new part of the message.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_22114]{DRAFT} ⌈

Kind:	function	
Symbol:	Update(std::uint8_t in)	
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx	
Syntax:	`virtual ara::core::Result<void> Update (std::uint8_t in) noexcept=0;`	
Parameters (in):	in	a byte value that is a part of input message
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Started	if the digest calculation was not initiated by a call of the Start() method
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"	
Description:	Update the digest calculation context by a new part of the message. This method is convenient for processing of constant tags.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_22210]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Description:	Get ExtensionService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_22213]{DRAFT} ⌈

Kind:	function	
Symbol:	GetMaxInputSize(bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx	
Syntax:	`virtual std::size_t GetMaxInputSize (bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	suppressPadding	if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only
Return value:	std::size_t	maximum size of the input data block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Description:	Get maximum expected size of the input data block. If (IsEncryption() == false) then a value returned by this method is independent from the suppressPadding argument and it will be equal to the block size.	

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_22214]{DRAFT} ⌈

Kind:	function	
Symbol:	GetMaxOutputSize(bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx	
Syntax:	`virtual std::size_t GetMaxOutputSize (bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	suppressPadding	if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only
Return value:	std::size_t	maximum size of the output data block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Description:	Get maximum possible size of the output data block. If (IsEncryption() == true) then a value returned by this method is independent from the suppressPadding argument and will be equal to the block size.	

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_22215]{DRAFT} ⌈

Kind:	function	
Symbol:	DecodeAndVerify(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > DecodeAndVerify (ReadOnlyMemRegion in) const noexcept=0;`	
Parameters (in):	in	the input data block

▽

△

Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	the output buffer actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInput Size	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutput Size(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppress Padding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_22216]{DRAFT} ⌈

Kind:	function	
Symbol:	DecodeAndVerify(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > DecodeAndVerify (ReadOnlyMem Region in) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	the input data block
Return value:	ara::core::Result< ByteVector< Alloc > >	the managed container for output block
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInput Size	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_22212]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_22211]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_22511]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPublicKey()	
Scope:	class ara::crypto::cryp::PrivateKey	
Syntax:	`virtual ara::core::Result<PublicKey::Uptrc> GetPublicKey () const noexcept=0;`	
Return value:	ara::core::Result< PublicKey::Uptrc >	unique smart pointer to the public key correspondent to this private key
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/private_key.h"	
Description:	Get the public key correspondent to this private key.	

⌋*(RS_CRYPTO_02108, RS_CRYPTO_02115)*

[SWS_CRYPT_22711]{DRAFT} ⌈

Kind:	function	
Symbol:	CheckKey(bool strongCheck=true)	
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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"	
Description:	Check the key for its correctness.	

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_22712]{DRAFT} ⌈

Kind:	function	
Symbol:	HashPublicKey(HashFunctionCtx &hashFunc)	
Scope:	class ara::crypto::cryp::PublicKey	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > HashPublicKey (HashFunctionCtx &hashFunc) const noexcept=0;`	
Parameters (in):	hashFunc	a hash-function instance that should be used the hashing
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	a buffer preallocated for the resulting hash value
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficientCapacity	if size of the hash buffer is not enough for storing of the result
	SecurityErrorDomain::kIncompleteArgState	if the hashFunc context is not initialized
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"	
Description:	Calculate hash of the Public Key value. The original public key value BLOB is available via the Serializable interface.	

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_22713]{DRAFT} ⌈

Kind:	function
Symbol:	HashPublicKey(HashFunctionCtx &hashFunc)
Scope:	class ara::crypto::cryp::PublicKey
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > HashPublicKey (HashFunctionCtx &hashFunc) const noexcept;`

▽

△

Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	hashFunc	a hash-function instance that should be used the hashing
Return value:	ara::core::Result< ByteVector< Alloc > >	pre-reserved managed container for the resulting hash value
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if capacity of the hash buffer is not enough for storing of the result
	SecurityErrorDomain::kIncompleteArg State	if the hashFunc context is not initialized
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"	
Description:	Calculate hash of the Public Key value. This method sets the size of the output container according to actually saved value! The original public key value BLOB is available via the Serializable interface.	

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_22914]{DRAFT} ⌈

Kind:	function	
Symbol:	AddEntropy(ReadOnlyMemRegion entropy)	
Scope:	class ara::crypto::cryp::RandomGeneratorCtx	
Syntax:	`virtual bool AddEntropy (ReadOnlyMemRegion entropy) noexcept=0;`	
Parameters (in):	entropy	a memory region with the additional entropy value
Return value:	bool	true if the method is supported and the entropy has been updated successfully
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Update the internal state of the RNG by mixing it with the provided additional entropy. This method is optional for implementation. An implementation of this method may "accumulate" provided entropy for future use.	

⌋(*RS_CRYPTO_02206*)

[SWS_CRYPT_22915]{DRAFT} ⌈

Kind:	function	
Symbol:	Generate(std::uint32_t count)	
Scope:	class ara::crypto::cryp::RandomGeneratorCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > Generate (std::uint32_t count) noexcept=0;`	
Parameters (in):	count	number of random bytes to generate
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	a buffer filled with the generated random sequence
Exception Safety:	noexcept	

▽

△

Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if this context implements a local RNG (i.e., the RNG state is controlled by the application), and has to be seeded by the application because it either has not already been seeded or ran out of entropy.
	SecurityErrorDomain::kBusyResource	if this context implements a global RNG (i.e., the RNG state is controlled by the stack and not the application) that is currently out-of-entropy and therefore cannot provide the requested number of random bytes
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Return an allocated buffer with a generated random sequence of the requested size.	

⌋*(RS_CRYPTO_02206)*

[SWS_CRYPT_22902]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::RandomGeneratorCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Get ExtensionService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_22911]{DRAFT} ⌈

Kind:	function	
Symbol:	Seed(ReadOnlyMemRegion seed)	
Scope:	class ara::crypto::cryp::RandomGeneratorCtx	
Syntax:	`virtual bool Seed (ReadOnlyMemRegion seed) noexcept=0;`	
Parameters (in):	seed	a memory region with the seed value
Return value:	bool	true if the method is supported and the state has been set successfully
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Set the internal state of the RNG using the provided seed.	

⌋*(RS_CRYPTO_02206)*

[SWS_CRYPT_22912]{DRAFT} ⌈

Kind:	function	
Symbol:	Seed(const SecretSeed &seed)	
Scope:	class ara::crypto::cryp::RandomGeneratorCtx	
Syntax:	`virtual bool Seed (const SecretSeed &seed) noexcept=0;`	
Parameters (in):	seed	a memory region with the seed value
Return value:	bool	true if the method is supported and the state has been set successfully
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Set the internal state of the RNG using the provided seed.	

⌋(*RS_CRYPTO_02206*)

[SWS_CRYPT_22913]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const SymmetricKey &key)	
Scope:	class ara::crypto::cryp::RandomGeneratorCtx	
Syntax:	`virtual bool SetKey (const SymmetricKey &key) noexcept=0;`	
Parameters (in):	key	a SymmetricKey with the key used as seed value
Return value:	bool	true if the method is supported and the key has been set successfully
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Set the internal state of the RNG using the provided seed.	

⌋(*RS_CRYPTO_02001*, *RS_CRYPTO_02003*)

[SWS_CRYPT_24811]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02008*)

[SWS_CRYPT_23011]{DRAFT} ⌈

Kind:	function	
Symbol:	Clone(ReadOnlyMemRegion xorDelta=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::SecretSeed	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptr> Clone (ReadOnlyMemRegion xorDelta=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	xorDelta	optional "delta" value that must be XOR-ed with the "cloned" copy of the original seed
Return value:	ara::core::Result< SecretSeed::Uptr >	unique smart pointer to "cloned" session Secret Seed object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Description:	Clone this Secret Seed object to new session object. Created object instance is session and non-exportable, AllowedUsageFlags attribute of the "cloned" object is identical to this attribute of the source object! If size of the xorDelta argument is less than the value size of this seed then only correspondent number of leading bytes of the original seed should be XOR-ed, but the rest should be copied without change. If size of the xorDelta argument is larger than the value size of this seed then extra bytes of the xorDelta should be ignored.	

⌋(*RS_CRYPTO_02007*)

[SWS_CRYPT_23012]{DRAFT} ⌈

Kind:	function	
Symbol:	JumpFrom(const SecretSeed &from, std::int64_t steps)	
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 >	reference to this updated object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if this object and the from argument are associated with incompatible cryptographic algorithms
	SecurityErrorDomain::kInvalidInputSize	if value size of the from seed is less then value size of this one
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Description:	Set value of this seed object as a "jump" from an initial state to specified number of steps, according to "counting" expression defined by a cryptographic algorithm associated with this object. steps may have positive and negative values that correspond to forward and backward direction of the "jump" respectively, but 0 value means only copy from value to this seed object. Seed size of the from argument always must be greater or equal of this seed size.	

⌋(*RS_CRYPTO_02007*)

[SWS_CRYPT_23014]{DRAFT} ⌈

Kind:	function	
Symbol:	Jump(std::int64_t steps)	
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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Description:	Set value of this seed object as a "jump" from it's current state to specified number of steps, according to "counting" expression defined by a cryptographic algorithm associated with this object. steps may have positive and negative values that correspond to forward and backward direction of the "jump" respectively, but 0 value means no changes of the current seed value.	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_23013]{DRAFT} ⌈

Kind:	function	
Symbol:	Next()	
Scope:	class ara::crypto::cryp::SecretSeed	
Syntax:	`virtual SecretSeed& Next () noexcept=0;`	
Return value:	SecretSeed &	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Description:	Set next value of the secret seed according to "counting" expression defined by a cryptographic algorithm associated with this object. If the associated cryptographic algorithm doesn't specify a "counting" expression then generic increment operation must be implemented as default (little-endian notation, i.e. first byte is least significant). .	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_23015]{DRAFT} ⌈

Kind:	function	
Symbol:	operator^=(const SecretSeed &source)	
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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	

▽

△

Description:	XOR value of this seed object with another one and save result to this object. If seed sizes in this object and in the source argument are different then only correspondent number of leading bytes in this seed object should be updated.

⌋(*RS_CRYPTO_02007*)

[SWS_CRYPT_23016]{DRAFT} ⌈

Kind:	function	
Symbol:	operator^=(ReadOnlyMemRegion source)	
Scope:	class ara::crypto::cryp::SecretSeed	
Syntax:	`virtual SecretSeed& operator^= (ReadOnlyMemRegion source) noexcept=0;`	
Parameters (in):	source	right argument for the XOR operation
Return value:	SecretSeed &	reference to this updated object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"	
Description:	XOR value of this seed object with provided memory region and save result to this object. If seed sizes in this object and in the source argument are different then only correspondent number of leading bytes of this seed object should be updated.	

⌋(*RS_CRYPTO_02007*)

[SWS_CRYPT_19906]{DRAFT} ⌈

Kind:	function	
Symbol:	SecurityException(ara::core::ErrorCode err)	
Scope:	class ara::crypto::SecurityException	
Syntax:	`explicit SecurityException (ara::core::ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/security_error_domain.h"	
Description:	Construct a new SecurityException from an ErrorCode.	

⌋(*SWS_CORE_10910*)

[SWS_CRYPT_23210]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–

▽

△

Exception Safety:	noexcept
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Description:	Extension service member class.

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_23213]{DRAFT} ⌈

Kind:	function	
Symbol:	GetMaxInputSize(bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx	
Syntax:	`virtual std::size_t GetMaxInputSize (bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	suppressPadding	if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only
Return value:	std::size_t	maximum size of the input data block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Description:	Get maximum expected size of the input data block. If (IsEncryption() == false) then a value returned by this method is independent from the suppressPadding argument and it will be equal to the block size.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23214]{DRAFT} ⌈

Kind:	function	
Symbol:	GetMaxOutputSize(bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx	
Syntax:	`virtual std::size_t GetMaxOutputSize (bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	suppressPadding	if true then the method calculates the size for the case when the whole space of the plain data block is used for the payload only
Return value:	std::size_t	maximum size of the output data block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Description:	Get maximum possible size of the output data block. If (IsEncryption() == true) then a value returned by this method is independent from the suppressPadding argument and will be equal to the block size.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23215]{DRAFT} ⌈

Kind:	function	
Symbol:	SignAndEncode(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > SignAndEncode (ReadOnlyMemRegion in) const noexcept=0;`	
Parameters (in):	in	the input data block
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	the output buffer actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInput Size	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutput Size(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppress Padding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_23216]{DRAFT} ⌈

Kind:	function	
Symbol:	SignAndEncode(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > SignAndEncode (ReadOnlyMemRegion in) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	the input data block
Return value:	ara::core::Result< ByteVector< Alloc > >	the managed container for output block
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInput Size	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	

▽

△

Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_23212]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_23211]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_23311]{DRAFT} ⌈

Kind:	function	
Symbol:	GetHashAlgId()	
Scope:	class ara::crypto::cryp::Signature	
Syntax:	`virtual CryptoPrimitiveId::AlgId GetHashAlgId () const noexcept=0;`	
Return value:	CryptoPrimitiveId::AlgId	ID of used hash algorithm only (without signature algorithm specification)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/cryobj/signature.h"	
Description:	Get an ID of hash algorithm used for this signature object production.	

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02203, RS_CRYPTO_02205)*

[SWS_CRYPT_23312]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29003]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRequiredHashAlgId()	
Scope:	class ara::crypto::cryp::SignatureService	
Syntax:	`virtual CryptoPrimitiveId::AlgId GetRequiredHashAlgId () const noexcept=0;`	
Return value:	CryptoPrimitiveId::AlgId	required hash algorithm ID or kAlgIdAny if the signature algorithm specification does not include a concrete hash function
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/signature_service.h"	
Description:	Get an ID of hash algorithm required by current signature algorithm.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_29002]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_29004]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02309*)

[SWS_CRYPT_23510]{DRAFT} ⌈

Kind:	function	
Symbol:	GetSignatureService()	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual SignatureService::Uptr GetSignatureService () const noexcept=0;`	
Return value:	SignatureService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Get SignatureService instance.	

⌋(*RS_CRYPTO_02006*)

[SWS_CRYPT_23516]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Clear the crypto context.	

⌋(*RS_CRYPTO_02108*)

[SWS_CRYPT_23515]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const PrivateKey &key)	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context is prohibited by /// the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Set (deploy) a key to the signer private algorithm context.	

⌋(*RS_CRYPTO_02001, RS_CRYPTO_02003*)

[SWS_CRYPT_23511]{DRAFT} ⌈

Kind:	function	
Symbol:	SignPreHashed(const HashFunctionCtx &hashFn, ReadOnlyMemRegion context=ReadOnly MemRegion())	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> SignPreHashed (const Hash FunctionCtx &hashFn, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	hashFn	a finalized hash-function context that contains a digest value ready for sign
	context	an optional user supplied "context" (its support depends from concrete algorithm)

▽

△

Return value:	ara::core::Result< Signature::Uptrc >	unique smart pointer to serialized signature
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if hash-function algorithm does not comply with the signature algorithm specification of this context
	SecurityErrorDomain::kInvalidInputSize	if the user supplied context has incorrect (or unsupported) size
	SecurityErrorDomain::kProcessingNot Finished	if the method hash.Finish() was not called before the call of this method
	SecurityErrorDomain::kUninitialized Context	this context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Sign a provided digest value stored in the hash-function context. This method must put the hash-function algorithm ID and a COUID of the used key-pair to the resulting signature object! The user supplied context may be used for such algorithms as: Ed25519ctx, Ed25519ph, Ed448ph. If the target algorithm doesn't support the context argument then the empty (default) value must be supplied!	

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_23512]{DRAFT} ⌈

Kind:	function	
Symbol:	Sign(ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > Sign (ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMem Region()) const noexcept=0;`	
Parameters (in):	value	the (pre-)hashed or direct message value that should be signed
	context	an optional user supplied "context" (its support depends from concrete algorithm)
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	actual size of the signature value stored to the output buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if size of the input value or context arguments are incorrect / unsupported
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Sign a directly provided hash or message value. This method can be used for implementation of the "multiple passes" signature algorithms that process a message directly, i.e. without "pre-hashing" (like Ed25519ctx). But also this method is suitable for implementation of the traditional signature schemes with pre-hashing (like Ed25519ph, Ed448ph, ECDSA). If the target algorithm doesn't support the context argument then the empty (default) value must be supplied!	

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_23513]{DRAFT} ⌈

Kind:	function	
Symbol:	SignPreHashed(AlgId hashAlgId, ReadOnlyMemRegion hashValue, ReadOnlyMemRegion context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> SignPreHashed (AlgId hashAlgId, ReadOnlyMemRegion hashValue, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	hashAlgId	hash function algorithm ID
	hashValue	hash function value (resulting digest without any truncations)
	context	an optional user supplied "context" (its support depends from concrete algorithm)
Return value:	ara::core::Result< Signature::Uptrc >	unique smart pointer to serialized signature
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if hash-function algorithm does not comply with the signature algorithm specification of this context
	SecurityErrorDomain::kInvalidInputSize	if the user supplied context has incorrect (or unsupported) size
	SecurityErrorDomain::kUninitializedContext	this context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Sign a directly provided digest value and create the Signature object. This method must put the hash-function algorithm ID and a COUID of the used key-pair to the resulting signature object! The user supplied context may be used for such algorithms as: Ed25519ctx, Ed25519ph, Ed448ph. If the target algorithm doesn't support the context argument then the empty (default) value must be supplied!	

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_23514]{DRAFT} ⌈

Kind:	function	
Symbol:	Sign(ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > Sign (ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	value	the (pre-)hashed or direct message value that should be signed
	context	an optional user supplied "context" (its support depends from concrete algorithm)
Return value:	ara::core::Result< ByteVector< Alloc > >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if size of the input value or context arguments are incorrect / unsupported

▽

△

	SecurityErrorDomain::kInsufficient Capacity	if capacity of the output signature container is not enough
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Sign a directly provided hash or message value. This method can be used for implementation of the "multiple passes" signature algorithms that process a message directly, i.e. without "pre-hashing" (like Ed25519ctx). But also this method is suitable for implementation of the traditional signature schemes with pre-hashing (like Ed25519ph, Ed448ph, ECDSA). This method sets the size of the output container according to actually saved value! If the target algorithm doesn't support the context argument then the empty (default) value must be supplied!	

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_23620]{DRAFT} ⌈

Kind:	function	
Symbol:	CountBytesInCache()	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual std::size_t CountBytesInCache () const noexcept=0;`	
Return value:	std::size_t	number of bytes now kept in the context cache
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Count number of bytes now kept in the context cache. In block-wise modes if an application has supplied input data chunks with incomplete last block then the context saves the rest part of the last (incomplete) block to internal "cache" memory and wait a next call for additional input to complete this block.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23621]{DRAFT} ⌈

Kind:	function	
Symbol:	EstimateMaxInputSize(std::size_t outputCapacity)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`std::size_t EstimateMaxInputSize (std::size_t outputCapacity) const noexcept;`	
Parameters (in):	outputCapacity	capacity of the output buffer
Return value:	std::size_t	maximum number of input bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Estimate maximal number of input bytes that may be processed for filling of an output buffer without overflow.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23622]{DRAFT} ⌈

Kind:	function	
Symbol:	EstimateRequiredCapacity(std::size_t inputSize, bool isFinal=false)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`std::size_t EstimateRequiredCapacity (std::size_t inputSize, bool isFinal=false) const noexcept;`	
Parameters (in):	inputSize	size of input data
	isFinal	flag that indicates processing of the last data chunk (if true)
Return value:	std::size_t	required capacity of the output buffer (in bytes)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Estimate minimal required capacity of the output buffer, which is enough for saving a result of input data processing.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23618]{DRAFT} ⌈

Kind:	function	
Symbol:	FinishBytes(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > FinishBytes (ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	an input data buffer
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	an output data buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficientCapacity	if capacity of the output buffer is not enough
	SecurityErrorDomain::kInOutBuffersIntersect	if the input and output buffers intersect
	SecurityErrorDomain::kProcessingNotStarted	if data processing was not started by a call of the Start() method
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Processe 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)! in an input data buffer an output data buffer SecurityErrorDomain::kInsufficientCapacity if capacity of the output buffer is not enough SecurityErrorDomain::kInOutBuffersIntersect if the input and output buffers intersect SecurityErrorDomain::kProcessingNotStarted if data processing was not started by a call ofthe Start() method	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23619]{DRAFT} ⌈

Kind:	function	
Symbol:	FinishBytes(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > FinishBytes (ReadOnlyMemRegion in) noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	an input data buffer The input buffer must not point inside the output container!
Return value:	ara::core::Result< ByteVector< Alloc > >	a managed container for output data
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if capacity of the output container is not enough
	SecurityErrorDomain::kInOutBuffers Intersect	if the input and output buffers intersect
	SecurityErrorDomain::kProcessingNot Started	if data processing was not started by a call of the Start() method
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Processe the final part of message (that may be not aligned to the block-size boundary). This method sets the size of the output container according to actually saved value. If (IsBytewise Mode() == false) then it must be: bs = GetBlockSize(), out.capacity() >= (((in.size() + bs * ((CryptoTransform::kEncrypt == GetTransformation.Value()) ? 2 : 1) - 1) / bs) * bs) If (Is BytewiseMode() == true) then it must be: out.capacity() >= in.size() Usage of this method is mandatory for processing of the last data chunk in block-wise modes! This method may be used for processing of a whole message in a single call (in any mode)!	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23602]{DRAFT} ⌈

Kind:	function	
Symbol:	GetBlockService()	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual BlockService::Uptr GetBlockService () const noexcept=0;`	
Return value:	BlockService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Get BlockService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_23611]{DRAFT} ⌈

Kind:	function	
Symbol:	IsBytewiseMode()	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual bool IsBytewiseMode () const noexcept=0;`	
Return value:	bool	true if the mode can process messages the byte-by-byte (without padding up to the block boundary) and false if only the block-by-block (only full blocks can be processed, the padding is mandatory)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Check the operation mode for the bytewise property.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23624]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23612]{DRAFT} ⌈

Kind:	function	
Symbol:	IsSeekableMode()	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Check if the seek operation is supported in the current mode.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23614]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlocks(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ProcessBlocks (ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	an input data buffer
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	an output data buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatibleArguments	if sizes of the input and output buffers are not equal
	SecurityErrorDomain::kInvalidInputSize	if size of the input buffer is not divisible by the block size (see GetBlockSize())
	SecurityErrorDomain::kInOutBuffersIntersect	if the input and output buffers partially intersect
	SecurityErrorDomain::kInvalidUsageOrder	if this method is called after processing of non-aligned data (to the block-size boundary)
	SecurityErrorDomain::kProcessingNotStarted	if the data processing was not started by a call of the Start() method
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Processe initial parts of message aligned to the block-size boundary. It is a copy-optimized method that doesn't use the internal cache buffer! It can be used only before processing of any non-aligned to the block-size boundary data. Pointers to the input and output buffers must be aligned to the block-size boundary! The input and output buffers may completely coincide, but they must not partially intersect!	

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_23615]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlocks(ReadWriteMemRegion inOut)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<void> ProcessBlocks (ReadWriteMemRegion inOut) noexcept=0;`	
Parameters (inout):	inOut	an input and output data buffer, i.e. the whole buffer should be updated
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if size of the inOut buffer is not divisible by the block size (see GetBlockSize())
	SecurityErrorDomain::kInvalidUsageOrder	if this method is called after processing of non-aligned data (to the block-size boundary)
	SecurityErrorDomain::kProcessingNotStarted	if the data processing was not started by a call of the Start() method
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	

▽

△

Description:	Processe initial parts of message aligned to the block-size boundary. It is a copy-optimized method that doesn't use internal cache buffer! It can be used up to first non-block aligned data processing. Pointer to the input-output buffer must be aligned to the block-size boundary!

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_23616]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBytes(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ProcessBytes (ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	an input data buffer
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	an output data buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if the output buffer has capacity insufficient for placing of the transformation result
	SecurityErrorDomain::kInOutBuffers Intersect	if the input and output buffers intersect
	SecurityErrorDomain::kProcessingNot Started	if data processing was not started by a call of the Start() method
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Processe a non-final part of message (that is not aligned to the block-size boundary). If (Is BytewiseMode() == false) then it must be: bs = GetBlockSize(), out.size() >= (((in.size() + bs - 1) / bs) * bs) If (IsBytewiseMode() == true) then it must be: out.size() >= in.size() The input and output buffers must not intersect! This method is "copy inefficient", therefore it should be used only in conditions when an application cannot control the chunking of the original message!	

⌋(*RS_CRYPTO_02302*)

[SWS_CRYPT_23617]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBytes(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ProcessBytes (ReadOnlyMemRegion in) noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	an input data buffer
Return value:	ara::core::Result< ByteVector< Alloc > >	a managed container for the output data
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if capacity of the output container is not enough

▽

△

	SecurityErrorDomain::kInOutBuffers Intersect	if the input buffer points inside of the preallocated output container
	:	SecurityErrorDomain::kProcessingNotStarted if data processing was not started by a call of the Start() method
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Processes a non-final part of message (that is not aligned to the block-size boundary). This method sets size of the output container according to actually saved value. If (IsBytewiseMode() == false) then it must be: bs = GetBlockSize(), out.capacity() >= (((in.size() + bs - 1) / bs) * bs) If (IsBytewiseMode() == true) then it must be: out.capacity() >= in.size() This method is "copy inefficient", therefore it should be used only in conditions when an application cannot control the chunking of the original message! The input buffer must not point inside the output container!	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23627]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_23613]{DRAFT} ⌈

Kind:	function	
Symbol:	Seek(std::int64_t offset, bool fromBegin=true)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<void> Seek (std::int64_t offset, bool fromBegin=true) noexcept=0;`	
Parameters (in):	offset	the offset value in bytes, relative to begin or current position in the gamma stream
	fromBegin	the starting point for positioning within the stream: from begin (if true) or from current position (if false)
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
	SecurityErrorDomain::kUnsupported	if the seek operation is not supported by the current mode
	SecurityErrorDomain::kProcessingNot Started	if the data processing was not started by a call of the Start() method

▽

△

	SecurityErrorDomain::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)
	SecurityErrorDomain::kInvalid Argument	if the offset is not aligned to the required boundary (see IsBytewiseMode())
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Set the position of the next byte within the stream of the encryption/decryption gamma.	

⌋(*RS_CRYPTO_02304*)

[SWS_CRYPT_23623]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt) noexcept=0;`	
Parameters (in):	key	the source key object
DIRECTION NOT DEFINED	transform	–
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Set (deploy) a key to the stream chiper algorithm context.	

⌋(*RS_CRYPTO_02001, RS_CRYPTO_02003*)

[SWS_CRYPT_23625]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(ReadOnlyMemRegion iv=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<void> Start (ReadOnlyMemRegion iv=ReadOnly MemRegion()) noexcept=0;`	
Parameters (in):	iv	an optional Initialization Vector (IV) or "nonce" value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by deploying a key

▽

△

	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation, but provided IV value is not empty, i.e. if (iv.empty() == false)
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23626]{DRAFT} ⌈

Kind:	function	
Symbol:	Start(const SecretSeed &iv)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<void> Start (const SecretSeed &iv) noexcept=0;`	
Parameters (in):	iv	the Initialization Vector (IV) or "nonce" object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by deploying a key
	SecurityErrorDomain::kInvalidInputSize	if the size of provided IV is not supported (i.e. if it is not enough for the initialization)
	SecurityErrorDomain::kUnsupported	if the base algorithm (or its current implementation) principally doesn't support the IV variation
	SecurityErrorDomain::kUsageViolation	if this transformation type is prohibited by the "allowed usage" restrictions of the provided Secret Seed object
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Initialize the context for a new data stream processing or generation (depending from the primitive). If IV size is greater than maximally supported by the algorithm then an implementation may use the leading bytes only from the sequence.	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23702]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCryptoService()	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`virtual CryptoService::Uptr GetCryptoService () const noexcept=0;`	
Return value:	CryptoService::Uptr	–

▽

△

Exception Safety:	noexcept
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"
Description:	Get CryptoService instance.

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_23711]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23712]{DRAFT} ⌈

Kind:	function	
Symbol:	IsMaxInputOnly()	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`ara::core::Result<bool> IsMaxInputOnly () const noexcept;`	
Return value:	ara::core::Result< bool >	true if the transformation requires the maximum size of input data and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Indicate that the currently configured transformation accepts only complete blocks of input data.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23713]{DRAFT} ⌈

Kind:	function	
Symbol:	IsMaxOutputOnly()	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`ara::core::Result<bool> IsMaxOutputOnly () const noexcept;`	
Return value:	ara::core::Result< bool >	true if the transformation can produce only the maximum size of output data and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the transformation direction of this context is configurable during an initialization, but the context was not initialized yet
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Indicate that the currently configured transformation can produce only complete blocks of output data.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23716]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ProcessBlock (ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept=0;`	
Parameters (in):	in	the input data block
	suppressPadding	if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	the output buffer Encryption with (suppressPadding == true) expects that: in.size() == GetMaxInputSize(true) && out.size() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects that: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.size() >= GetMaxOutputSize(false). Decryption expects that: in.size() == GetMaxInputSize() && out.size() >= GetMaxOutputSize(suppressPadding). The case (out.size() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! actual size of output data (it always <= out.size()) or 0 if the input data block has incorrect content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
	:	SecurityErrorDomain::kIncorrectInputSize if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kInsufficient Capacity	if the out.size() is not enough to store the transformation result

▽

△

	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration.	

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_23717]{DRAFT} ⌈

Kind:	function	
Symbol:	ProcessBlock(ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ProcessBlock (ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	the input data block
	suppressPadding	if true then the method doesn't apply the padding, but the payload should fill the whole block of the plain data
Return value:	ara::core::Result< ByteVector< Alloc > >	the managed container for output block
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncorrectInput Size	if the mentioned above rules about the input size is violated
	SecurityErrorDomain::kInsufficient Capacity	if the out.size() is not enough to store the transformation result
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration. This method sets the size of the output container according to actually saved value! Encryption with (suppressPadding == true) expects what: in.size() == GetMaxInputSize(true) && out.capacity() >= GetMaxOutputSize(true). Encryption with (suppressPadding == false) expects what: in.size() <= GetMaxInputSize(false) && in.size() > 0 && out.capacity() >= GetMaxOutput Size(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= Get MaxOutputSize(suppressPadding). The case (out.capacity() < GetMaxOutputSize()) should be used with caution, only if you are strictly certain about the size of the output data! In case of (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)!	

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_23715]{DRAFT} ⌈

Kind:	function
Symbol:	ProcessBlocks(ReadOnlyMemRegion in)
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx

▽

△

Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ProcessBlocks (ReadOnlyMemRegion in) const noexcept=0;`	
Parameters (in):	in	an input data buffer
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	an output data buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
	SecurityErrorDomain::kInvalidInputSize	if size of the input buffer is not divisible by the block size (see GetBlockSize())
	SecurityErrorDomain::kIncompatible Arguments	if sizes of the input and output buffer are not equal
	SecurityErrorDomain::kInOutBuffers Intersect	if the input and output buffers partially intersect
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Processe provided blocks without padding. The in and out buffers must have same size and this size must be divisible by the block size (see GetBlockSize()). Pointers to the input and output buffers must be aligned to the block-size boundary!	

⌋*(RS_CRYPTO_02302)*

[SWS_CRYPT_23714]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_23710]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt)	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const SymmetricKey &key, CryptoTransform transform=CryptoTransform::kEncrypt) noexcept=0;`	
Parameters (in):	key	the source key object
DIRECTION NOT DEFINED	transform	–
Return value:	ara::core::Result< void >	–

▽

△

Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Set (deploy) a key to the symmetric algorithm context.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_24013]{DRAFT} ⌈

Kind:	function	
Symbol:	CalculateWrappedKeySize(std::size_t keyLength)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual std::size_t CalculateWrappedKeySize (std::size_t keyLength) const noexcept=0;`	
Parameters (in):	keyLength	original key length in bits
Return value:	std::size_t	size of the wrapped key in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Calculate size of the wrapped key in bytes from original key length in bits. This method can be useful for some implementations different from RFC3394 / RFC5649.	

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_24002]{DRAFT} ⌈

Kind:	function	
Symbol:	GetExtensionService()	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ExtensionService::Uptr GetExtensionService () const noexcept=0;`	
Return value:	ExtensionService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Get ExtensionService instance.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_24012]{DRAFT} ⌈

Kind:	function	
Symbol:	GetMaxTargetKeyLength()	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual std::size_t GetMaxTargetKeyLength () const noexcept=0;`	
Return value:	std::size_t	maximum length of the target key in bits
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Get maximum length of the target key supported by the implementation. This method can be useful for some implementations different from RFC3394 / RFC5649.	

⌋(*RS_CRYPTO_02201*)

[SWS_CRYPT_24011]{DRAFT} ⌈

Kind:	function	
Symbol:	GetTargetKeyGranularity()	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual std::size_t GetTargetKeyGranularity () const noexcept=0;`	
Return value:	std::size_t	size of the block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Get expected granularity of the target key (block size). If the class implements RFC3394 (KW without padding) then this method should return 8 (i.e. 8 octets = 64 bits). If the class implements RFC5649 (KW with padding) then this method should return 1 (i.e. 1 octet = 8 bits).	

⌋(*RS_CRYPTO_02201*)

[SWS_CRYPT_24019]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Clear the crypto context.	

⌋(*RS_CRYPTO_02108*)

[SWS_CRYPT_24018]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const SymmetricKey &key, CryptoTransform transform)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const SymmetricKey &key, CryptoTransform transform) noexcept=0;`	
Parameters (in):	key	the source key object
DIRECTION NOT DEFINED	transform	–
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context (taking into account the direction specified by transform) is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Set (deploy) a key to the symmetric key wrapper algorithm context.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_24017]{DRAFT} ⌈

Kind:	function	
Symbol:	UnwrapConcreteKey(ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`template <typename ExpectedKey> ara::core::Result<typename ExpectedKey::Uptrc> UnwrapConcreteKey (ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage) noexcept;`	
Template param:	ExpectedKey	the expected type of concrete key
Parameters (in):	wrappedKey	a memory region that contains wrapped key
	algId	an identifier of the target symmetric crypto algorithm
	allowedUsage	bit-flags that define a list of allowed transformations' types in which the target key can be used
Return value:	ara::core::Result< typename ExpectedKey::Uptrc >	unique smart pointer to ExpectedKey object, which keeps unwrapped key material
Exception Safety:	noexcept	
Errors:	SecurityErrorDomain::kInvalidInputSize	if the size of provided wrapped key is unsupported
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Execute the "key unwrap" operation for provided BLOB and produce a Key object of expected type. For additional details see UnwrapKey()	

⌋*(RS_CRYPTO_02115)*

[SWS_CRYPT_24016]{DRAFT} ⌈

Kind:	function	
Symbol:	UnwrapKey(ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowed Usage)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<RestrictedUseObject::Uptrc> UnwrapKey (Read OnlyMemRegion 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if the size of provided wrapped key is unsupported
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Execute the "key unwrap" operation for provided BLOB and produce Key object. This method should be compliant to RFC3394 or RFC5649, if implementation is based on the AES block cipher and applied to an AES key. The created Key object has following attributes: session and non-exportable (because it was imported without meta-information)! SymmetricKey may be unwrapped in following way: SymmetricKey::Uptrc key = SymmetricKey::Cast(Unwrap Key(wrappedKey, ...)); PrivateKey may be unwrapped in following way: PrivateKey::Uptrc key = PrivateKey::Cast(UnwrapKey(wrappedKey, ...)); In both examples the Cast() method may additionally throw the BadObjectTypeException if an actual type of the unwrapped key differs from the target one!	

⌋*(RS_CRYPTO_02115)*

[SWS_CRYPT_24015]{DRAFT} ⌈

Kind:	function	
Symbol:	UnwrapSeed(ReadOnlyMemRegion wrappedSeed, AlgId targetAlgId, SecretSeed::Usage allowedUsage)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> UnwrapSeed (ReadOnlyMem Region wrappedSeed, AlgId targetAlgId, SecretSeed::Usage allowedUsage) const noexcept=0;`	
Parameters (in):	wrappedSeed	a memory region that contains wrapped seed
	targetAlgId	the target symmetric algorithm identifier (also defines a target seed-length)
	allowedUsage	allowed usage scope of the target seed
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to SecretSeed object, which keeps unwrapped key material
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidInputSize	if the size of provided wrapped seed is unsupported

▽

△

	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Execute the "key unwrap" operation for provided BLOB and produce SecretSeed object. This method should be compliant to RFC3394 or RFC5649, if implementation is based on the AES block cipher and applied to an AES key material. The created SecretSeed object has following attributes: session and non-exportable (because it was imported without meta-information).	

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_24014]{DRAFT} ⌈

Kind:	function	
Symbol:	WrapKeyMaterial(const RestrictedUseObject &key)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > WrapKeyMaterial (const RestrictedUseObject &key) const noexcept=0;`	
Parameters (in):	key	a key that should be wrapped
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if the size of the wrapped buffer is not enough for storing the result
	SecurityErrorDomain::kInvalidInputSize	if the key object has an unsupported length
	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Execute the "key wrap" operation for the provided key material. This method should be compliant to RFC3394 or RFC5649, if an implementation is based on the AES block cipher and applied to an AES key. Method CalculateWrappedKeySize() can be used for size calculation of the required output buffer.	

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_24102]{DRAFT} ⌈

Kind:	function	
Symbol:	GetSignatureService()	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	`virtual SignatureService::Uptr GetSignatureService () const noexcept=0;`	
Return value:	SignatureService::Uptr	–
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Extension service member class.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_24116]{DRAFT} ⌈

Kind:	function	
Symbol:	Reset()	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	`virtual ara::core::Result<void> Reset () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Clear the crypto context.	

⌋*(RS_CRYPTO_02108)*

[SWS_CRYPT_24115]{DRAFT} ⌈

Kind:	function	
Symbol:	SetKey(const PublicKey &key)	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	`virtual ara::core::Result<void> SetKey (const PublicKey &key) noexcept=0;`	
Parameters (in):	key	the source key object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the provided key object is incompatible with this symmetric key context
	SecurityErrorDomain::kUsageViolation	if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Set (deploy) a key to the verifier public algorithm context.	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

[SWS_CRYPT_24111]{DRAFT} ⌈

Kind:	function	
Symbol:	VerifyPrehashed(const HashFunctionCtx &hashFn, const Signature &signature, ReadOnlyMem Region context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	`ara::core::Result<bool> VerifyPrehashed (const HashFunctionCtx &hash Fn, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMem Region()) const noexcept;`	
Parameters (in):	hashFn	hash function to be used for hashing
	signature	the signature object for verification
	context	an optional user supplied "context" (its support depends from concrete algorithm)

▽

△

Return value:	ara::core::Result< bool >	true if the signature was verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNot Finished	if the method hashFn.Finish() was not called before this method call
	SecurityErrorDomain::kInvalid Argument	if the CryptoAlgId of hashFn differs from the Crypto AlgId of this context
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Verify signature by a digest value stored in the hash-function context. This is a pass-through interface to SWS_CRYPT_24113 for developer convenience, i.e. it adds additional input checks amd then calls the verify() interface from SWS_CRYPT_24113.	

⌋(*RS_CRYPTO_02204*)

[SWS_CRYPT_24112]{DRAFT} ⌈

Kind:	function	
Symbol:	Verify(ReadOnlyMemRegion value, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	`virtual ara::core::Result<bool> Verify (ReadOnlyMemRegion value, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	value	the (pre-)hashed or direct message value that should be verified
	signature	the signature BLOB for the verification (the BLOB contains a plain sequence of the digital signature components located in fixed/maximum length fields defined by the algorithm specification, and each component is presented by a raw bytes sequence padded by zeroes to full length of the field; e.g. in case of (EC)DSA-256 (i.e. length of the q module is 256 bits) the signature BLOB must have two fixed-size fields: 32 + 32 bytes, for R and S components respectively, i.e. total BLOB size is 64 bytes)
	context	an optional user supplied "context" (its support depends from concrete algorithm)
Return value:	ara::core::Result< bool >	true if the signature was verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUninitialized Context	if the context was not initialized by a key value
	SecurityErrorDomain::kInvalidInputSize	if the context argument has unsupported size
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	

▽

△

Description:	Verify signature BLOB by a directly provided hash or message value. This method can be used for implementation of the "multiple passes" signature algorithms that process a message directly, i.e. without "pre-hashing" (like Ed25519ctx). But also this method is suitable for implementation of the traditional signature schemes with pre-hashing (like Ed25519ph, Ed448ph, ECDSA). If the target algorithm doesn't support the context argument then the empty (default) value must be supplied! The user supplied context may be used for such algorithms as: Ed25519ctx, Ed25519ph, Ed448ph.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_24113]{DRAFT} ⌈

Kind:	function	
Symbol:	VerifyPrehashed(CryptoAlgId hashAlgId, ReadOnlyMemRegion hashValue, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	`ara::core::Result<bool> VerifyPrehashed (CryptoAlgId hashAlgId, ReadOnlyMemRegion hashValue, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept;`	
Parameters (in):	hashAlgId	hash function algorithm ID
	hashValue	hash function value (resulting digest without any truncations)
	signature	the signature object for the verification
	context	an optional user supplied "context" (its support depends from concrete algorithm)
Return value:	ara::core::Result< bool >	true if the signature was verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the CryptoAlgId of this context does not match the CryptoAlgId of signature; or the required CryptoAlg Id of the hash is not kAlgIdDefault and the required hash CryptoAlgId of this context does not match hashAlgId or the hash CryptoAlgId of signature
	SecurityErrorDomain::kIncompatible Arguments	if the provided hashAlgId is not kAlgIdDefault and the AlgId of the provided signature object does not match the provided hashAlgId
	SecurityErrorDomain::kBadObject Reference	if the provided signature object does not reference the public key loaded to the context, i.e. if the COUID of the public key in the context is not equal to the COUID referenced from the signature object.
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Verify signature by a digest value stored in the hash-function context. This is a pass-through interface to SWS_CRYPT_24112 for developer convenience, i.e. it adds additional input checks amd then calls the default verify() interface.	

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_24114]{DRAFT} ⌈

Kind:	function	
Symbol:	VerifyPrehashed(const HashFunctionCtx &hashFn, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	virtual ara::core::Result<bool> VerifyPrehashed (const HashFunctionCtx &hashFn, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept;	
Parameters (in):	hashFn	hash function to be used for hashing
	signature	the data BLOB to be verified
	context	an optional user supplied "context" (its support depends from concrete algorithm)
Return value:	ara::core::Result< bool >	true if the signature was verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kProcessingNotFinished	if the method hashFn.Finish() was not called before this method call
	SecurityErrorDomain::kInvalidArgument	if the CryptoAlgId of hashFn differs from the CryptoAlgId of this context
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Verify signature by a digest value stored in the hash-function context. This is a pass-through interface to SWS_CRYPT_24112 for developer convenience, i.e. it adds additional input checks amd then calls the default verify() interface.	

⌋ *(RS_CRYPTO_02204)*

[SWS_CRYPT_24101]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::VerifierPublicCtx
Derived from:	std::unique_ptr<VerifierPublicCtx>
Syntax:	using Uptr = std::unique_ptr<VerifierPublicCtx>;
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"
Description:	Unique smart pointer of the interface.

⌋ *(RS_CRYPTO_02204)*

[SWS_CRYPT_20101]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::AuthCipherCtx
Derived from:	std::unique_ptr<AuthCipherCtx>
Syntax:	using Uptr = std::unique_ptr<AuthCipherCtx>;
Header file:	#include "ara/crypto/cryp/auth_cipher_ctx.h"

▽

△

Description:	Unique smart pointer of the interface.

⌋(RS_CRYPTO_02207)

[SWS_CRYPT_24802]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::RestrictedUseObject
Derived from:	std::unique_ptr<RestrictedUseObject>
Syntax:	`using Uptrc = std::unique_ptr<RestrictedUseObject>;`
Header file:	#include "ara/crypto/cryp/cryobj/restricted_use_object.h"
Description:	Unique smart pointer of the interface.

⌋(RS_CRYPTO_02403)

[SWS_CRYPT_29031]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::BlockService
Derived from:	std::unique_ptr<BlockService>
Syntax:	`using Uptr = std::unique_ptr<BlockService>;`
Header file:	#include "ara/crypto/cryp/block_service.h"
Description:	Unique smart pointer of the interface.

⌋(RS_CRYPTO_02309)

[SWS_CRYPT_20402]{DRAFT} ⌈

Kind:	type alias
Symbol:	AlgId
Scope:	class ara::crypto::cryp::CryptoContext
Derived from:	CryptoAlgId
Syntax:	`using AlgId = CryptoAlgId;`
Header file:	#include "ara/crypto/cryp/crypto_context.h"
Description:	Type definition of vendor specific binary Crypto Primitive ID.

⌋(RS_CRYPTO_02008)

[SWS_CRYPT_20504]{DRAFT} ⌈

Kind:	struct
Symbol:	COIdentifier
Scope:	class ara::crypto::cryp::CryptoObject
Syntax:	`struct COIdentifier {...};`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Description:	Unique identifier of this CryptoObject.

⌋(*RS_CRYPTO_02005*)

[SWS_CRYPT_20502]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::CryptoObject
Derived from:	std::unique_ptr<const CryptoObject>
Syntax:	`using Uptrc = std::unique_ptr<const CryptoObject>;`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Description:	Unique smart pointer of the constant interface.

⌋(*RS_CRYPTO_02005*)

[SWS_CRYPT_20501]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::CryptoObject
Derived from:	std::unique_ptr<CryptoObject>
Syntax:	`using Uptr = std::unique_ptr<CryptoObject>;`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02005*)

[SWS_CRYPT_20641]{DRAFT} ⌈

Kind:	type alias
Symbol:	AlgId
Scope:	class ara::crypto::cryp::CryptoPrimitiveId
Derived from:	CryptoAlgId
Syntax:	`using AlgId = CryptoAlgId;`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Description:	Type definition of vendor specific binary Crypto Primitive ID.

⌋(*RS_CRYPTO_02005*)

[SWS_CRYPT_20644]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::CryptoPrimitiveId
Derived from:	std::unique_ptr<const CryptoPrimitiveId>
Syntax:	`using Uptrc = std::unique_ptr<const CryptoPrimitiveId>;`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Description:	type definition pointer

⌋(*RS_CRYPTO_02005*)

[SWS_CRYPT_20643]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::CryptoPrimitiveId
Derived from:	std::unique_ptr<CryptoPrimitiveId>
Syntax:	`using Uptr = std::unique_ptr<CryptoPrimitiveId>;`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_primitive_id.h"
Description:	type definition pointer to const

⌋(*RS_CRYPTO_02005*)

[SWS_CRYPT_20703]{DRAFT} ⌈

Kind:	type alias
Symbol:	AlgId
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	CryptoPrimitiveId::AlgId
Syntax:	`using AlgId = CryptoPrimitiveId::AlgId;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	A short alias for Algorithm ID type definition.

⌋(*RS_CRYPTO_02005, RS_CRYPTO_02006*)

[SWS_CRYPT_20701]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	std::unique_ptr<CryptoProvider>
Syntax:	`using Uptr = std::unique_ptr<CryptoProvider>;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Shared smart pointer of the interface.

⌋(*RS_CRYPTO_02109*)

[SWS_CRYPT_29024]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::CryptoService
Derived from:	std::unique_ptr<CryptoService>
Syntax:	`using Uptr = std::unique_ptr<CryptoService>;`
Header file:	#include "ara/crypto/cryp/crypto_service.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_20801]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::DecryptorPrivateCtx
Derived from:	std::unique_ptr<DecryptorPrivateCtx>
Syntax:	`using Uptr = std::unique_ptr<DecryptorPrivateCtx>;`
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_29011]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::DigestService
Derived from:	std::unique_ptr<DigestService>
Syntax:	`using Uptr = std::unique_ptr<DigestService>;`
Header file:	#include "ara/crypto/cryp/digest_service.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_21001]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::EncryptorPublicCtx
Derived from:	std::unique_ptr<EncryptorPublicCtx>
Syntax:	`using Uptr = std::unique_ptr<EncryptorPublicCtx>;`
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"

▽

△

Description:	Unique smart pointer of the interface.

⌋(RS_CRYPTO_02202)

[SWS_CRYPT_29042]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::ExtensionService
Derived from:	std::unique_ptr<ExtensionService>
Syntax:	`using Uptr = std::unique_ptr<ExtensionService>;`
Header file:	#include "ara/crypto/cryp/extension_service.h"
Description:	Unique smart pointer of the interface.

⌋(RS_CRYPTO_02309)

[SWS_CRYPT_21101]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::HashFunctionCtx
Derived from:	std::unique_ptr<HashFunctionCtx>
Syntax:	`using Uptr = std::unique_ptr<HashFunctionCtx>;`
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"
Description:	Unique smart pointer of the interface.

⌋(RS_CRYPTO_02205)

[SWS_CRYPT_21301]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx
Derived from:	std::unique_ptr<KeyAgreementPrivateCtx>
Syntax:	`using Uptr = std::unique_ptr<KeyAgreementPrivateCtx>;`
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"
Description:	Unique smart pointer of this interface.

⌋(RS_CRYPTO_02104)

[SWS_CRYPT_21401]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx
Derived from:	std::unique_ptr<KeyDecapsulatorPrivateCtx>
Syntax:	`using Uptr = std::unique_ptr<KeyDecapsulatorPrivateCtx>;`
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02104*)

[SWS_CRYPT_21501]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx
Derived from:	std::unique_ptr<KeyDerivationFunctionCtx>
Syntax:	`using Uptr = std::unique_ptr<KeyDerivationFunctionCtx>;`
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02103*)

[SWS_CRYPT_21801]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx
Derived from:	std::unique_ptr<KeyEncapsulatorPublicCtx>
Syntax:	`using Uptr = std::unique_ptr<KeyEncapsulatorPublicCtx>;`
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02209*)

[SWS_CRYPT_22101]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx
Derived from:	std::unique_ptr<MessageAuthnCodeCtx>
Syntax:	`using Uptr = std::unique_ptr<MessageAuthnCodeCtx>;`
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02203*)

[SWS_CRYPT_22201]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx
Derived from:	std::unique_ptr<MsgRecoveryPublicCtx>
Syntax:	`using Uptr = std::unique_ptr<MsgRecoveryPublicCtx>;`
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_22501]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::PrivateKey
Derived from:	std::unique_ptr<const PrivateKey>
Syntax:	`using Uptrc = std::unique_ptr<const PrivateKey>;`
Header file:	#include "ara/crypto/cryp/cryobj/private_key.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_22701]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::PublicKey
Derived from:	std::unique_ptr<const PublicKey>
Syntax:	`using Uptrc = std::unique_ptr<const PublicKey>;`
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02202)*

[SWS_CRYPT_22901]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::RandomGeneratorCtx
Derived from:	std::unique_ptr<RandomGeneratorCtx>
Syntax:	`using Uptr = std::unique_ptr<RandomGeneratorCtx>;`
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"

▽

△

Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02206)*

[SWS_CRYPT_24801]{DRAFT} ⌈

Kind:	type alias
Symbol:	Usage
Scope:	class ara::crypto::cryp::RestrictedUseObject
Derived from:	AllowedUsageFlags
Syntax:	`using Usage = AllowedUsageFlags;`
Header file:	#include "ara/crypto/cryp/cryobj/restricted_use_object.h"
Description:	Alias to the container type for bit-flags of allowed usages of the object.

⌋*(RS_CRYPTO_02008)*

[SWS_CRYPT_23001]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::SecretSeed
Derived from:	std::unique_ptr<const SecretSeed>
Syntax:	`using Uptrc = std::unique_ptr<const SecretSeed>;`
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Description:	Unique smart pointer of a constant interface instance.

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_23002]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::SecretSeed
Derived from:	std::unique_ptr<SecretSeed>
Syntax:	`using Uptr = std::unique_ptr<SecretSeed>;`
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Description:	Unique smart pointer of a volatile interface instance.

⌋*(RS_CRYPTO_02007)*

[SWS_CRYPT_23201]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx
Derived from:	std::unique_ptr<SigEncodePrivateCtx>
Syntax:	`using Uptr = std::unique_ptr<SigEncodePrivateCtx>;`
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02202)*

[SWS_CRYPT_29001]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::SignatureService
Derived from:	std::unique_ptr<SignatureService>
Syntax:	`using Uptr = std::unique_ptr<SignatureService>;`
Header file:	#include "ara/crypto/cryp/signature_service.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_23301]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::Signature
Derived from:	std::unique_ptr<const Signature>
Syntax:	`using Uptrc = std::unique_ptr<const Signature>;`
Header file:	#include "ara/crypto/cryp/cryobj/signature.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02203, RS_CRYPTO_02205)*

[SWS_CRYPT_23501]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::SignerPrivateCtx
Derived from:	std::unique_ptr<SignerPrivateCtx>
Syntax:	`using Uptr = std::unique_ptr<SignerPrivateCtx>;`
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02204)*

[SWS_CRYPT_23601]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::StreamCipherCtx
Derived from:	std::unique_ptr<StreamCipherCtx>
Syntax:	`using Uptr = std::unique_ptr<StreamCipherCtx>;`
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_23701]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx
Derived from:	std::unique_ptr<SymmetricBlockCipherCtx>
Syntax:	`using Uptr = std::unique_ptr<SymmetricBlockCipherCtx>;`
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_23801]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::cryp::SymmetricKey
Derived from:	std::unique_ptr<const SymmetricKey>
Syntax:	`using Uptrc = std::unique_ptr<const SymmetricKey>;`
Header file:	#include "ara/crypto/cryp/cryobj/symmetric_key.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02201)*

[SWS_CRYPT_24001]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx
Derived from:	std::unique_ptr<SymmetricKeyWrapperCtx>
Syntax:	`using Uptr = std::unique_ptr<SymmetricKeyWrapperCtx>;`
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"

▽

△

Description:	Unique smart pointer of the interface.

⌋(RS_CRYPTO_02201)

[SWS_CRYPT_20506]{DRAFT} ⌈

Kind:	variable
Symbol:	mCOType
Scope:	struct ara::crypto::cryp::CryptoObject::COIdentifier
Type:	CryptoObjectType
Syntax:	`CryptoObjectType mCOType;`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Description:	type of objext

⌋(RS_CRYPTO_02005)

[SWS_CRYPT_20507]{DRAFT} ⌈

Kind:	variable
Symbol:	mCouid
Scope:	struct ara::crypto::cryp::CryptoObject::COIdentifier
Type:	CryptoObjectUid
Syntax:	`CryptoObjectUid mCouid;`
Header file:	#include "ara/crypto/cryp/cryobj/crypto_object.h"
Description:	object identifier

⌋(RS_CRYPTO_02005)

[SWS_CRYPT_22503]{DRAFT} ⌈

Kind:	variable
Symbol:	kObjectType
Scope:	class ara::crypto::cryp::PrivateKey
Type:	const CryptoObjectType
Syntax:	`const CryptoObjectType kObjectType = CryptoObjectType::kPrivateKey;`
Header file:	#include "ara/crypto/cryp/cryobj/private_key.h"
Description:	Static mapping of this interface to specific value of CryptoObjectType enumeration.

⌋(RS_CRYPTO_02306)

[SWS_CRYPT_22702]{DRAFT} ⌈

Kind:	variable
Symbol:	kObjectType
Scope:	class ara::crypto::cryp::PublicKey
Type:	const CryptoObjectType
Syntax:	`const CryptoObjectType kObjectType = CryptoObjectType::kPublicKey;`
Header file:	#include "ara/crypto/cryp/cryobj/public_key.h"
Description:	const object type

⌋(*RS_CRYPTO_02202*)

[SWS_CRYPT_23003]{DRAFT} ⌈

Kind:	variable
Symbol:	kObjectType
Scope:	class ara::crypto::cryp::SecretSeed
Type:	const CryptoObjectType
Syntax:	`const CryptoObjectType kObjectType = CryptoObjectType::kSecretSeed;`
Header file:	#include "ara/crypto/cryp/cryobj/secret_seed.h"
Description:	Static mapping of this interface to specific value of CryptoObjectType enumeration.

⌋(*RS_CRYPTO_02007*)

[SWS_CRYPT_23302]{DRAFT} ⌈

Kind:	variable
Symbol:	kObjectType
Scope:	class ara::crypto::cryp::Signature
Type:	const CryptoObjectType
Syntax:	`const CryptoObjectType kObjectType = CryptoObjectType::kSignature;`
Header file:	#include "ara/crypto/cryp/cryobj/signature.h"
Description:	Signature object initialized.

⌋(*RS_CRYPTO_02204, RS_CRYPTO_02203, RS_CRYPTO_02205*)

[SWS_CRYPT_23802]{DRAFT} ⌈

Kind:	variable
Symbol:	kObjectType
Scope:	class ara::crypto::cryp::SymmetricKey
Type:	const CryptoObjectType
Syntax:	`const CryptoObjectType kObjectType = CryptoObjectType::kSymmetricKey;`
Header file:	#include "ara/crypto/cryp/cryobj/symmetric_key.h"
Description:	const object type

⌋(*RS_CRYPTO_02201*)

[SWS_CRYPT_10101]{DRAFT} ⌈

Kind:	variable
Symbol:	mGeneratorUid
Scope:	struct ara::crypto::CryptoObjectUid
Type:	Uuid
Syntax:	`Uuid mGeneratorUid;`
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Description:	UUID of a generator that has produced this COUID. This UUID can be associated with HSM, physical host/ECU or VM.

⌋(*RS_CRYPTO_02006*)

8.2 C++ language binding Key Storage Provider

[SWS_CRYPT_30400]{DRAFT} ⌈

Kind:	class
Symbol:	KeySlot
Scope:	namespace ara::crypto::keys
Syntax:	`class KeySlot {...};`
Header file:	#include "ara/crypto/keys/keyslot.h"
Description:	Key slot port-prototype interface. This class enables access to a physicl key-slot.

⌋(*RS_CRYPTO_02405*)

[SWS_CRYPT_30100]{DRAFT} ⌈

Kind:	class
Symbol:	KeyStorageProvider
Scope:	namespace ara::crypto::keys
Syntax:	`class KeyStorageProvider {...};`
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	Key Storage Provider interface. Any object is uniquely identified by the combination of its UUID and type. HSMs/TPMs implementing the concept of "non-extractable keys" should use own copies of externally supplied crypto objects. A few software Crypto Providers can share single key slot if they support same format.

⌋(*RS_CRYPTO_02109*, *RS_CRYPTO_02305*, *RS_CRYPTO_02401*)

[SWS_CRYPT_30200]{DRAFT} ⌈

Kind:	class
Symbol:	UpdatesObserver
Scope:	namespace ara::crypto::keys
Syntax:	`class UpdatesObserver {...};`
Header file:	#include "ara/crypto/keys/updates_observer.h"
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.

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30405]{DRAFT} ⌈

Kind:	function	
Symbol:	Clear()	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`virtual ara::core::Result<void> Clear () noexcept=0;`	
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnreserved Resource	if the target slot is not opened writeable.
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Clear the content of this key-slot. This method must perform a secure cleanup without the ability to restore the object data! This method may be used for atomic update of a key slot scoped to some transaction. In such case the the slot will be updated only after correspondent call of CommitTransaction().	

⌋*(RS_CRYPTO_02009)*

[SWS_CRYPT_30510]{DRAFT} ⌈

Kind:	function
Symbol:	KeySlotContentProps()
Scope:	struct ara::crypto::keys::KeySlotContentProps
Syntax:	`KeySlotContentProps ()=default;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	set content properties

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30401]{DRAFT} ⌈

Kind:	function	
Symbol:	~KeySlot()	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`virtual ~KeySlot () noexcept=default;`	
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Destructor.	

⌋(*RS_CRYPTO_02405*)

[SWS_CRYPT_30408]{DRAFT} ⌈

Kind:	function	
Symbol:	GetContentProps()	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`virtual ara::core::Result<KeySlotContentProps> GetContentProps () const noexcept=0;`	
Return value:	ara::core::Result< KeySlotContent Props >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kEmpty Container	if the slot is empty
	SecurityErrorDomain::kAccess Violation	if this method is called by an Actor, which has no any ("Owner" or "User") access rights to the key slot
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Get an actual properties of a content in the key slot. If this method called by a "User" Actor then always: props.exportability == false.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30403]{DRAFT} ⌈

Kind:	function	
Symbol:	MyProvider()	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`virtual ara::core::Result<cryp::CryptoProvider::Uptr> MyProvider () const noexcept=0;`	
Return value:	ara::core::Result< cryp::Crypto Provider::Uptr >	a unique_pointer to the CryptoProvider to be used with this KeySlot
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/keyslot.h"	

▽

△

Description:	Retrieve an instance of the CryptoProvider that owns this KeySlot. Any key slot always has an associated default Crypto Provider that can serve this key slot. In the simplest case all key slots can be served by a single Crypto Provider installed on the Adaptive Platform. But in a more complicated case a few different Crypto Providers may coexist in the system, for example if ECU has one or a few HSMs and software cryptography implementation too, and each of them has own physical key storage. In such case different dedicated Crypto Providers may serve mentioned HSMs and the software implementation. .

⌋*(RS_CRYPTO_02401)*

[SWS_CRYPT_30407]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPrototypedProps()	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`virtual ara::core::Result<KeySlotPrototypeProps> GetPrototypedProps () const noexcept=0;`	
Return value:	ara::core::Result< KeySlotPrototype Props >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Get the prototyped properties of the key slot.	

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30404]{DRAFT} ⌈

Kind:	function	
Symbol:	IsEmpty()	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`virtual bool IsEmpty () const noexcept=0;`	
Return value:	bool	true if the slot is empty or false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Check the slot for emptiness.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30409]{DRAFT} ⌈

Kind:	function
Symbol:	Open(bool subscribeForUpdates=false, bool writeable=false)
Scope:	class ara::crypto::keys::KeySlot

▽

△

Syntax:	`virtual ara::core::Result<IOInterface::Uptr> Open (bool subscribeForUpdates=false, bool writeable=false) const noexcept=0;`	
Parameters (in):	subscribeForUpdates	if this flag is true then the UpdatesObserver instance (previously registered by a call of the method RegisterObserver()) will be subscribed for updates of the opened key slot
	writeable	indicates whether the key-slot shall be opened read-only (default) or with write access
Return value:	ara::core::Result< IOInterface::Uptr >	an unique smart pointer to the IOInterface associated with the slot content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidUsageOrder	if (true == subscribeForUpdates), but there is no registered instance of the UpdatesObserver in the Key Storage Provider context
	SecurityErrorDomain::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
	SecurityErrorDomain::kModifiedResource	if the specified slot has been modified after the Key Slot has been opened
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Open this key slot and return an IOInterface to its content. If the UpdatesObserver interface was provided to the call of RegisterObserver() then the UpdatesObserver::OnUpdate() method should be called by Key Storage engine (in a dedicated thread) every time when this slot is updated (and become visible for "Users"). Monitoring of the opened key slot will be continued even after destruction of the returned TrustedContainer, because content of the slot may be loaded to volatile memory (as a CryptoObject or to a CryptoContext of a crypto primitive), but the TrustedContainer may be destroyed after this. Therefore if you need to terminate monitoring of the key slot then you should directly call method UnsubscribeObserver(SlotNumber).	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30301]{DRAFT} ⌈

Kind:	function
Symbol:	KeySlotPrototypeProps()
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Syntax:	`KeySlotPrototypeProps ()=default;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	

⌋(*RS_CRYPTO_02110*)

[SWS_CRYPT_30406]{DRAFT} ⌈

Kind:	function
Symbol:	SaveCopy(const IOInterface &container)
Scope:	class ara::crypto::keys::KeySlot

▽

△

Syntax:	`virtual ara::core::Result<void> SaveCopy (const IOInterface &container) noexcept=0;`	
Parameters (in):	container	the source IOInterface
Return value:	ara::core::Result< void >	true if successfully saved
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompatible Object	if the source object has property "session" or if the source IOInterface references a KeySlot from a different CryptoProvider
	SecurityErrorDomain::kEmpty Container	if the source IOInterface is empty
	SecurityErrorDomain::kContent Restrictions	if the source object doesn't satisfy the slot restrictions (including version control)
	SecurityErrorDomain::kUnreserved Resource	if the target slot is not opened writeable.
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Save the content of a provided source IOInterface to this key-slot. The source container may represent a volatile trusted container or another KeySlot This method may be used for atomic update of a key slot scoped to some transaction. In such case the the slot will be updated only after correspondent call of CommitTransaction().	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30220]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const KeySlot &other)	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`KeySlot& operator= (const KeySlot &other)=default;`	
Parameters (in):	other	the other instance
Return value:	KeySlot &	*this, containing the contents of other
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Copy-assign another KeySlot to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30221]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(KeySlot &&other)	
Scope:	class ara::crypto::keys::KeySlot	
Syntax:	`KeySlot& operator= (KeySlot &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	KeySlot &	*this, containing the contents of other
Header file:	#include "ara/crypto/keys/keyslot.h"	
Description:	Move-assign another KeySlot to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30123]{DRAFT} ⌈

Kind:	function	
Symbol:	BeginTransaction(const TransactionScope &targetSlots)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<TransactionId> BeginTransaction (const TransactionScope &targetSlots) noexcept=0;`	
Parameters (in):	targetSlots	a list of KeySlots that should be updated during this transaction.
Return value:	ara::core::Result< TransactionId >	a unique ID assigned to this transaction
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnreserved Resource	if targetSlots list has a slot that has not been configured with the reserveSpareSlot parameter in the manifest
	SecurityErrorDomain::kBusyResource	if targetSlots list has key slots that are already involved to another pending transaction or opened in writing mode
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Begin new transaction for key slots update. In order for a keyslot to be part of a transaction scope, the reserveSpareSlot model parameter of the keyslot has to be set to true. A transaction is dedicated for updating related key slots simultaneously (in an atomic, all-or-nothing, way). All key slots that should be updated by the transaction have to be opened and provided to this function. Any changes to the slots in scope are executed by calling commit().	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30124]{DRAFT} ⌈

Kind:	function	
Symbol:	CommitTransaction(TransactionId id)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> CommitTransaction (TransactionId id) noexcept=0;`	
Parameters (in):	id	an ID of a transaction that should be commited
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back)
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Commit changes of the transaction to Key Storage. Any changes of key slots made during a transaction are invisible up to the commit execution. The commit command permanently saves all changes made during the transaction in Key Storage.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30110]{DRAFT} ⌈

Kind:	function	
Symbol:	~KeyStorageProvider()	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ~KeyStorageProvider () noexcept=default;`	
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Destructor.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30131]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRegisteredObserver()	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual UpdatesObserver::Uptr GetRegisteredObserver () const noexcept=0;`	
Return value:	UpdatesObserver::Uptr	unique pointer to the registered Updates Observer interface (copy of an internal unique pointer is returned, i.e. the Key Storage provider continues to keep the ownership)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Get pointer of registered Updates Observer. The method returns nullptr if no observers have been registered yet!	

⌋*(RS_CRYPTO_02401)*

[SWS_CRYPT_30115]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadKeySlot(ara::core::InstanceSpecifier &iSpecify)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<KeySlot::Uptr> LoadKeySlot (ara::core::InstanceSpecifier &iSpecify) noexcept=0;`	
Parameters (in):	iSpecify	the target key-slot instance specifier
Return value:	ara::core::Result< KeySlot::Uptr >	an unique smart pointer to allocated key slot
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnreserved Resource	if the InstanceSpecifier is incorrect (the slot is not allocated)
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Load a key slot. The functions loads the information associated with a KeySlot into a KeySlot object.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30130]{DRAFT} ⌈

Kind:	function	
Symbol:	RegisterObserver(UpdatesObserver::Uptr observer=nullptr)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual UpdatesObserver::Uptr RegisterObserver (UpdatesObserver::Uptr observer=nullptr) noexcept=0;`	
Parameters (in):	observer	optional pointer to a client-supplied Updates Observer instance that should be registered inside Key Storage implementation and called every time, when an opened for usage/loading key slot is updated externally (by its "Owner" application)
Return value:	UpdatesObserver::Uptr	unique pointer to previously registered Updates Observer interface (the pointer ownership is "moved out" to the caller code)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Register consumer Updates Observer. Only one instance of the UpdatesObserver may be registered by an application process, therefore this method always unregister previous observer and return its unique pointer. If (nullptr == observer) then the method only unregister the previous observer! The method returns nullptr if no observers have been registered yet!	

⌋*(RS_CRYPTO_02401)*

[SWS_CRYPT_30125]{DRAFT} ⌈

Kind:	function	
Symbol:	RollbackTransaction(TransactionId id)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> RollbackTransaction (TransactionId id) noexcept=0;`	
Parameters (in):	id	an ID of a transaction that should be rolled back
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back)
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Rollback all changes executed during the transaction in Key Storage. The rollback command permanently cancels all changes made during the transaction in Key Storage. A rolled back transaction is completely invisible for all applications.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30126]{DRAFT} ⌈

Kind:	function	
Symbol:	UnsubscribeObserver(KeySlot &slot)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> UnsubscribeObserver (KeySlot &slot) noexcept=0;`	
Parameters (in):	slot	number of a slot that should be unsubscribed from the updates observing
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the specified slot is not monitored now (i.e. if it was not successfully opened via OpenAsUser() or it was already unsubscribed by this method)
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Unsubscribe the Update Observer from changes monitoring of the specified slot.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30222]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const KeyStorageProvider &other)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`KeyStorageProvider& operator= (const KeyStorageProvider &other)=default;`	
Parameters (in):	other	the other instance
Return value:	KeyStorageProvider &	*this, containing the contents of other
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Copy-assign another KeyStorageProvider to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30223]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(KeyStorageProvider &&other)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`KeyStorageProvider& operator= (KeyStorageProvider &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	KeyStorageProvider &	*this, containing the contents of other
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Move-assign another KeyStorageProvider to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30350]{DRAFT} ⌈

Kind:	function	
Symbol:	operator==(const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`constexpr bool operator== (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if all members' values of lhs is equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"	
Description:	Comparison operator "equal" for KeySlotPrototypeProps operands.	

⌋(*RS_CRYPTO_02110*)

[SWS_CRYPT_30351]{DRAFT} ⌈

Kind:	function	
Symbol:	operator!=(const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`constexpr bool operator!= (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if at least one member of lhs has a value not equal to correspondent member of rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"	
Description:	Comparison operator "not equal" for KeySlotPrototypeProps operands.	

⌋(*RS_CRYPTO_02110*)

[SWS_CRYPT_30550]{DRAFT} ⌈

Kind:	function	
Symbol:	operator==(const KeySlotContentProps &lhs, const KeySlotContentProps &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`constexpr bool operator== (const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if all members' values of lhs is equal to rhs, and false otherwise

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Comparison operator "equal" for KeySlotContentProps operands.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30551]{DRAFT} ⌈

Kind:	function	
Symbol:	operator!=(const KeySlotContentProps &lhs, const KeySlotContentProps &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`constexpr bool 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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"	
Description:	Comparison operator "not equal" for KeySlotContentProps operands.	

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30210]{DRAFT} ⌈

Kind:	function
Symbol:	~UpdatesObserver()
Scope:	class ara::crypto::keys::UpdatesObserver
Syntax:	`virtual ~UpdatesObserver () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/keys/updates_observer.h"
Description:	Destructor.

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30211]{DRAFT} ⌈

Kind:	function
Symbol:	OnUpdate(const TransactionScope &updatedSlots)
Scope:	class ara::crypto::keys::UpdatesObserver
Syntax:	`virtual void OnUpdate (const TransactionScope &updatedSlots) noexcept=0;`

▽

△

Parameters (in):	updatedSlots	List of monitored slots that were updated after opening (for reading)
Return value:	None	
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/updates_observer.h"	
Description:	Notification method that should be called if content of specified slots was changed. Key Storage engine should call this method in a dedicated thread. The provided list may include only slots subscribed for observing (during openning with the "User" permissions, i.e. for "reading" via a call of the method OpenAsUser()). Each slot number may present in the provided list only one time!	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30224]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const UpdatesObserver &other)	
Scope:	class ara::crypto::keys::UpdatesObserver	
Syntax:	`UpdatesObserver& operator= (const UpdatesObserver &other)=default;`	
Parameters (in):	other	the other instance
Return value:	UpdatesObserver &	*this, containing the contents of other
Header file:	#include "ara/crypto/keys/updates_observer.h"	
Description:	Copy-assign another UpdatesObserver to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30225]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(UpdatesObserver &&other)	
Scope:	class ara::crypto::keys::UpdatesObserver	
Syntax:	`UpdatesObserver& operator= (UpdatesObserver &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	UpdatesObserver &	*this, containing the contents of other
Header file:	#include "ara/crypto/keys/updates_observer.h"	
Description:	Move-assign another UpdatesObserver to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30500]{DRAFT} ⌈

Kind:	struct
Symbol:	KeySlotContentProps
Scope:	namespace ara::crypto::keys
Syntax:	`struct KeySlotContentProps {...};`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Properties of current Key Slot Content, i.e. of a current instance stored to the Key Slot. A value of the mAllowedUsage field is bitwise AND of the common usage flags defined at run-time and the usage flags defined by the UserPermissions prototype for current "Actor".

⌋(*RS_CRYPTO_02005, RS_CRYPTO_02111*)

[SWS_CRYPT_30511]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	struct ara::crypto::keys::KeySlotContentProps
Derived from:	std::unique_ptr<KeySlotContentProps>
Syntax:	`using Uptr = std::unique_ptr<KeySlotContentProps>;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	shared pointer of interface

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_30300]{DRAFT} ⌈

Kind:	struct
Symbol:	KeySlotPrototypeProps
Scope:	namespace ara::crypto::keys
Syntax:	`struct KeySlotPrototypeProps {...};`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Prototyped Properties of a Key Slot.

⌋(*RS_CRYPTO_02009, RS_CRYPTO_02110, RS_CRYPTO_02116*)

[SWS_CRYPT_30302]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Derived from:	std::unique_ptr<KeySlotPrototypeProps>
Syntax:	`using Uptr = std::unique_ptr<KeySlotPrototypeProps>;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	

⌋(*RS_CRYPTO_02110*)

[SWS_CRYPT_30402]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::keys::KeySlot
Derived from:	std::unique_ptr<KeySlot>
Syntax:	`using Uptr = std::unique_ptr<KeySlot>;`
Header file:	#include "ara/crypto/keys/keyslot.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02405*)

[SWS_CRYPT_30101]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::keys::KeyStorageProvider
Derived from:	std::unique_ptr<KeyStorageProvider>
Syntax:	`using Uptr = std::unique_ptr<KeyStorageProvider>;`
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30010]{DRAFT} ⌈

Kind:	type alias
Symbol:	TransactionId
Scope:	namespace ara::crypto::keys
Derived from:	std::uint64_t
Syntax:	`using TransactionId = std::uint64_t;`
Header file:	#include "ara/crypto/keys/elementary_types.h"
Description:	Definition of a transaction identifier type. The zero value should be reserved for especial cases.

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30011]{DRAFT} ⌈

Kind:	type alias
Symbol:	TransactionScope
Scope:	namespace ara::crypto::keys
Derived from:	ara::core::Vector<KeySlot>
Syntax:	`using TransactionScope = ara::core::Vector<KeySlot>;`
Header file:	#include "ara/crypto/keys/elementary_types.h"
Description:	Definition of a "transaction scope" type. The "transaction scope" defines a list of key slots that are target for update in a transaction.

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30201]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::keys::UpdatesObserver
Derived from:	std::unique_ptr<UpdatesObserver>
Syntax:	`using Uptr = std::unique_ptr<UpdatesObserver>;`
Header file:	#include "ara/crypto/keys/updates_observer.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30503]{DRAFT} ⌈

Kind:	variable
Symbol:	mAlgId
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	CryptoAlgId
Syntax:	`CryptoAlgId mAlgId;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Cryptoalgorithm of actual object stored to the slot.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30505]{DRAFT} ⌈

Kind:	variable
Symbol:	mObjectSize
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	std::size_t
Syntax:	`std::size_t mObjectSize;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Actual size of an object currently stored to the slot.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30508]{DRAFT} ⌈

Kind:	variable
Symbol:	mObjectType
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	CryptoObjectType
Syntax:	`CryptoObjectType mObjectType;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"

▽

△

Description:	Actual type of an object stored to the slot.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30501]{DRAFT} ⌈

Kind:	variable
Symbol:	mObjectUid
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	CryptoObjectUid
Syntax:	`CryptoObjectUid mObjectUid;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	UID of a Crypto Object stored to the slot.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30506]{DRAFT} ⌈

Kind:	variable
Symbol:	mContentAllowedUsage
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	AllowedUsageFlags
Syntax:	`AllowedUsageFlags mContentAllowedUsage;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Actual usage restriction flags of an object stored to the slot for the current "Actor".

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_30306]{DRAFT} ⌈

Kind:	variable
Symbol:	mAlgId
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	CryptoAlgId
Syntax:	`CryptoAlgId mAlgId;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Cryptoalgorithm restriction (kAlgIdAny means without restriction). The algorithm can be specified partially: family & length, mode, padding.

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30309]{DRAFT} ⌈

Kind:	variable
Symbol:	mAllocateSpareSlot
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	bool
Syntax:	`bool mAllocateSpareSlot;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Indicates whether FC Crypto shall allocate sufficient storage space for a shadow copy of this KeySlot.

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30310]{DRAFT} ⌈

Kind:	variable
Symbol:	mAllowContentTypeChange
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	bool
Syntax:	`bool mAllowContentTypeChange;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Indicates whether the content of this key-slot may be changed, e.g. from storing a symmetric key to storing an RSA key If this is set to false, then the mObjectType of this KeySlotPrototype Props must be a) valid and b) cannot be changed (i.e. only objects of mObjectType may be stored in this key-slot).

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30313]{DRAFT} ⌈

Kind:	variable
Symbol:	mContentAllowedUsage
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	AllowedUsageFlags
Syntax:	`AllowedUsageFlags mContentAllowedUsage;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Indicates how the content may be used. The following use cases of this attribute are considered: the object to be stored in this key-slot has it's AllowedUsageFlags set to kAllowPrototypedOnly. In this case this attribute must be observed when loading the content into a runtime instance (e.g. the AllowedUsageFlags of a SymmetricKey object should be set according to this attribute) mMaxUpdatesAllowed==0, in this case the content is provided during production while the AllowedUsageFlags is modeled using this attribute when this key-slot is flexibly updated the runtime object's AllowedUsageFlags override this attribute upon a later loading from this key-slot

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30312]{DRAFT} ⌈

Kind:	variable
Symbol:	mExportAllowed
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	bool
Syntax:	`bool mExportAllowed;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Indicates whether the key-slot content may be exported.

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30311]{DRAFT} ⌈

Kind:	variable
Symbol:	mMaxUpdateAllowed
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	std::int32_t
Syntax:	`std::int32_t mMaxUpdateAllowed;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Specifies how many times this key-slot may be updated, e.g.: a value of 0 means the key-slot content will be pre-set during production a value of 1 means the key-slot content can be updated only once ("OTP") a negative value means the key-slot content can be updated inifinitely

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30305]{DRAFT} ⌈

Kind:	variable
Symbol:	mSlotType
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	KeySlotType
Syntax:	`KeySlotType mSlotType;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Key-slot type configuration: all key-slots used by the adaptive machine to provide serives such as secure communication, diagnostics, updates, secure storage etc. shall use the type k Machine. All key-slots that will be used by the adaptive user application must use kApplication. A key-manager user application may define kMachine key-slots as well; in this case the integrator must match a corresponding machine key-slot to be managed.

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30307]{DRAFT} ⌈

Kind:	variable
Symbol:	mSlotCapacity
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	std::size_t
Syntax:	`std::size_t mSlotCapacity;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Capacity of the slot in bytes.

⌋*(RS_CRYPTO_02110)*

[SWS_CRYPT_30308]{DRAFT} ⌈

Kind:	variable
Symbol:	mObjectType
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	CryptoObjectType
Syntax:	`CryptoObjectType mObjectType;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Restriction of an object type that can be stored the slot. If this field contains CryptoObject Type::kUnknown then without restriction of the type.

⌋*(RS_CRYPTO_02110)*

8.3 C++ language binding X509 Certificate Management Provider

[SWS_CRYPT_40100]{DRAFT} ⌈

Kind:	class
Symbol:	BasicCertInfo
Scope:	namespace ara::crypto::x509
Base class:	X509Object
Syntax:	`class BasicCertInfo : public X509Object {...};`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	Basic Certificate Information interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40200]{DRAFT} ⌈

Kind:	class
Symbol:	Certificate
Scope:	namespace ara::crypto::x509
Base class:	BasicCertInfo
Syntax:	`class Certificate : public BasicCertInfo {...};`
Header file:	#include "ara/crypto/x509/certificate.h"
Description:	X.509 Certificate interface.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40300]{DRAFT} ⌈

Kind:	class
Symbol:	CertSignRequest
Scope:	namespace ara::crypto::x509
Base class:	BasicCertInfo
Syntax:	`class CertSignRequest : public BasicCertInfo {...};`
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Description:	Certificate Signing Request (CSR) object interface This interface is dedicated for complete parsing of the request content.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40700]{DRAFT} ⌈

Kind:	class
Symbol:	OcspRequest
Scope:	namespace ara::crypto::x509
Base class:	X509Object
Syntax:	`class OcspRequest : public X509Object {...};`
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Description:	On-line Certificate Status Protocol Request.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40800]{DRAFT} ⌈

Kind:	class
Symbol:	OcspResponse
Scope:	namespace ara::crypto::x509
Base class:	X509Object
Syntax:	`class OcspResponse : public X509Object {...};`
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Description:	On-line Certificate Status Protocol Response.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_24400]{DRAFT} ⌈

Kind:	class
Symbol:	X509PublicKeyInfo
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::Serializable
Syntax:	`class X509PublicKeyInfo : public Serializable {...};`
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"
Description:	X.509 Public Key Information interface.

⌋*(RS_CRYPTO_02307)*

[SWS_CRYPT_40400]{DRAFT} ⌈

Kind:	class
Symbol:	X509DN
Scope:	namespace ara::crypto::x509
Base class:	X509Object
Syntax:	`class X509DN : public X509Object {...};`
Header file:	#include "ara/crypto/x509/x509_dn.h"
Description:	Interface of X.509 Distinguished Name (DN).

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40500]{DRAFT} ⌈

Kind:	class
Symbol:	X509Extensions
Scope:	namespace ara::crypto::x509
Base class:	X509Object
Syntax:	`class X509Extensions : public X509Object {...};`
Header file:	#include "ara/crypto/x509/x509_extensions.h"
Description:	Interface of X.509 Extensions.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40900]{DRAFT} ⌈

Kind:	class
Symbol:	X509Object
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::Serializable
Syntax:	`class X509Object : public Serializable {...};`
Header file:	#include "ara/crypto/x509/x509_object.h"

▽

△

Description:	Common interface of all objects created by X.509 Provider.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40600]{DRAFT} ⌈

Kind:	class
Symbol:	X509Provider
Scope:	namespace ara::crypto::x509
Syntax:	`class X509Provider {...};`
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	X.509 Provider interface. The X.509 Provider supports two internal storages: volatile (or session) and persistent. All X.509 objects created by the provider should have an actual reference to their parent X.509 Provider. The X.509 Provider can be destroyed only after destroying of all its daughterly objects. Each method of this interface that creates a X.509 object is non-constant, because any such creation increases a references counter of the X.509 Provider.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_24414]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPublicKey()	
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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Description:	Get public key object of the subject. Created PublicKey object is session and non-exportable, because generic X.509 certificate or certificate signing request (CSR) doesn't have COUID of the public key, therefore it should be saved or transmitted only as a part of correspondent certificate or CSR.	

⌋(*RS_CRYPTO_02108, RS_CRYPTO_02306*)

[SWS_CRYPT_24412]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRequiredHashAlgId()	
Scope:	class ara::crypto::x509::X509PublicKeyInfo	
Syntax:	`virtual CryptoAlgId GetRequiredHashAlgId () const noexcept=0;`	
Return value:	CryptoAlgId	required hash algorithm ID or kAlgIdAny if the signature algorithm specification does not include a concrete hash function

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"
Description:	Get an ID of hash algorithm required by current signature algorithm.

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_24411]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRequiredHashSize()	
Scope:	class ara::crypto::x509::X509PublicKeyInfo	
Syntax:	`virtual std::size_t GetRequiredHashSize () const noexcept=0;`	
Return value:	std::size_t	required hash size in bytes
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Description:	Get the hash size required by current signature algorithm.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_24413]{DRAFT} ⌈

Kind:	function	
Symbol:	GetSignatureSize()	
Scope:	class ara::crypto::x509::X509PublicKeyInfo	
Syntax:	`virtual std::size_t GetSignatureSize () const noexcept=0;`	
Return value:	std::size_t	size of the signature value in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Description:	Get size of the signature value produced and required by the current algorithm.	

⌋*(RS_CRYPTO_02309)*

[SWS_CRYPT_24410]{DRAFT} ⌈

Kind:	function	
Symbol:	GetAlgorithmId()	
Scope:	class ara::crypto::x509::X509PublicKeyInfo	
Syntax:	`virtual ara::crypto::cryp::CryptoPrimitiveId::Uptrc GetAlgorithmId ()=0;`	
Return value:	ara::crypto::cryp::CryptoPrimitive Id::Uptrc	–
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	

▽

△

Description:	Get the CryptoPrimitiveId instance of this class.

⌋(*RS_CRYPTO_02307*)

[SWS_CRYPT_24415]{DRAFT} ⌈

Kind:	function	
Symbol:	IsSameKey(const ara::crypto::cryp::PublicKey &publicKey)	
Scope:	class ara::crypto::x509::X509PublicKeyInfo	
Syntax:	`virtual bool IsSameKey (const ara::crypto::cryp::PublicKey &publicKey) const noexcept=0;`	
Parameters (in):	publicKey	the public key object for comparison
Return value:	bool	true if values of the stored public key and object provided by the argument are identical and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"	
Description:	Verify the sameness of the provided and kept public keys. This method compare the public key values only.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40115]{DRAFT} ⌈

Kind:	function	
Symbol:	GetConstraints()	
Scope:	class ara::crypto::x509::BasicCertInfo	
Syntax:	`virtual KeyConstraints GetConstraints () const noexcept=0;`	
Return value:	KeyConstraints	key constraints
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Description:	Get the key constraints for the key associated with this PKCS#10 object.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40114]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPathLimit()	
Scope:	class ara::crypto::x509::BasicCertInfo	
Syntax:	`virtual std::uint32_t GetPathLimit () const noexcept=0;`	
Return value:	std::uint32_t	certification path length limit

▽

△

Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Description:	Get the constraint on the path length defined in the Basic Constraints extension.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40113]{DRAFT} ⌈

Kind:	function	
Symbol:	IsCa()	
Scope:	class ara::crypto::x509::BasicCertInfo	
Syntax:	`virtual bool IsCa () const noexcept=0;`	
Return value:	bool	true if it is a CA request and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Description:	Check whether the CA attribute of X509v3 Basic Constraints is true (i.e. pathlen=0).	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40112]{DRAFT} ⌈

Kind:	function	
Symbol:	SubjectDn()	
Scope:	class ara::crypto::x509::BasicCertInfo	
Syntax:	`virtual const X509DN& SubjectDn () const noexcept=0;`	
Return value:	const X509DN &	subject DN
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Description:	Get the subject DN.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40111]{DRAFT} ⌈

Kind:	function	
Symbol:	SubjectPubKey(cryp::CryptoProvider::Uptr cryptoProvider=nullptr)	
Scope:	class ara::crypto::x509::BasicCertInfo	
Syntax:	`virtual const X509PublicKeyInfo& SubjectPubKey (cryp::CryptoProvider::Uptr cryptoProvider=nullptr) const noexcept=0;`	
Parameters (in):	cryptoProvider	unique pointer of a target Crypto Provider, where the public key will be used

▽

△

Return value:	const X509PublicKeyInfo &	constant reference of the subject public key interface
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/basic_cert_info.h"	
Description:	Load the subject public key information object to realm of specified crypto provider. If (crypto Provider == nullptr) then X509PublicKeyInfo object will be loaded in realm of the Stack-default Crypto Provider.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40217]{DRAFT} ⌈

Kind:	function	
Symbol:	AuthorityKeyId()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > AuthorityKeyId () const noexcept=0;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	size of the DER encoded AuthorityKeyIdentifier in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if (id.empty() == false), but its size is not enough for storing the output value
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the DER encoded AuthorityKeyIdentifier of this certificate. If (id.empty() == true) then this method only returns required size of the output buffer.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40215]{DRAFT} ⌈

Kind:	function	
Symbol:	EndTime()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual time_t EndTime () const noexcept=0;`	
Return value:	time_t	"Not After" of the certificate
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the "Not After" of the certificate.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40220]{DRAFT} ⌈

Kind:	function	
Symbol:	GetFingerprint(ReadWriteMemRegion fingerprint, cryp::HashFunctionCtx &hashCtx)	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual ara::core::Result<std::size_t> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kIncompleteArgState	if the hashCtx context is not initialized
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Calculate a fingerprint from the whole certificate. The produced fingerprint value saved to the output buffer starting from leading bytes of the hash value. If the capacity of the output buffer is less than the digest size then the digest will be truncated and only leading bytes will be saved. If the capacity of the output buffer is higher than the digest size then only leading bytes of the buffer will be updated.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40221]{DRAFT} ⌈

Kind:	function	
Symbol:	GetStatus()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual Status GetStatus () const noexcept=0;`	
Return value:	Status	the certificate verification status
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Return last verification status of the certificate.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40212]{DRAFT} ⌈

Kind:	function	
Symbol:	IsRoot()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual bool IsRoot () const noexcept=0;`	
Return value:	bool	true if the TrustMaster has set this certificate as root
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Check whether this certificate belongs to a root CA.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40213]{DRAFT} ⌈

Kind:	function	
Symbol:	IssuerDn()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual const X509DN& IssuerDn () const =0;`	
Return value:	const X509DN &	Issuer DN of this certificate
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the issuer certificate DN.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40216]{DRAFT} ⌈

Kind:	function	
Symbol:	SerialNumber()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > SerialNumber () const noexcept=0;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	size of the certificate serial number in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficientCapacity	if (sn.empty() == false), but its size is not enough for storing the output value
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the serial number of this certificate. If (sn.empty() == true) then this method only returns required size of the output buffer.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40214]{DRAFT} ⌈

Kind:	function	
Symbol:	StartTime()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual time_t StartTime () const noexcept=0;`	
Return value:	time_t	"Not Before" of the certificate
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the "Not Before" of the certificate.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40218]{DRAFT} ⌈

Kind:	function	
Symbol:	SubjectKeyId()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > SubjectKeyId () const noexcept=0;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	size of the DER encoded SubjectKeyIdentifier in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if (id.empty() == false), but its size is not enough for storing the output value
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the DER encoded SubjectKeyIdentifier of this certificate. If (id.empty() == true) then this method only returns required size of the output buffer.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40219]{DRAFT} ⌈

Kind:	function	
Symbol:	VerifyMe(ara::core::Optional< const Certificate > caCert)	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual bool VerifyMe (ara::core::Optional< const Certificate > caCert) const noexcept=0;`	
Parameters (in):	caCert	the optional pointer to a Certification Authority certificate used for signature of the current one
Return value:	bool	true if this certificate was verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Verify signature of the certificate. Call with (caCert == nullptr) is applicable only if this is a certificate of a root CA.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40211]{DRAFT} ⌈

Kind:	function	
Symbol:	X509Version()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual std::uint32_t X509Version () const noexcept=0;`	
Return value:	std::uint32_t	X.509 version
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	

▽

△

Description:	Get the X.509 version of this certificate object.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40311]{DRAFT} ⌈

Kind:	function	
Symbol:	Verify()	
Scope:	class ara::crypto::x509::CertSignRequest	
Syntax:	`virtual bool Verify () const noexcept=0;`	
Return value:	bool	true if the signature is correct
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/cert_sign_request.h"	
Description:	Verifies self-signed signature of the certificate request.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40313]{DRAFT} ⌈

Kind:	function	
Symbol:	ExportASN1CertSignRequest()	
Scope:	class ara::crypto::x509::CertSignRequest	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ExportASN1CertSignRequest () noexcept=0;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	a buffer with the formatted CSR
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidUsageOrder	this error will be returned in case not all required information has been provided
Header file:	#include "ara/crypto/x509/cert_sign_request.h"	
Description:	Export this certificate signing request in DER encoded ASN1 format. Note: this is the CSR that can be sent to the CA for obtaining the certificate.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40315]{DRAFT} ⌈

Kind:	function
Symbol:	GetSignature()
Scope:	class ara::crypto::x509::CertSignRequest
Syntax:	`virtual const ara::crypto::cryp::Signature& GetSignature () const noexcept=0;`

▽

△

Return value:	const ara::crypto::cryp::Signature &	signature object of the request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/cert_sign_request.h"	
Description:	Return signature object of the request.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40314]{DRAFT} ⌈

Kind:	function	
Symbol:	Version()	
Scope:	class ara::crypto::x509::CertSignRequest	
Syntax:	`virtual unsigned Version () const noexcept=0;`	
Return value:	unsigned	format version of the certificate request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/cert_sign_request.h"	
Description:	Return format version of the certificate request.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40711]{DRAFT} ⌈

Kind:	function	
Symbol:	Version()	
Scope:	class ara::crypto::x509::OcspRequest	
Syntax:	`virtual std::uint32_t Version () const noexcept=0;`	
Return value:	std::uint32_t	OCSP request format version
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/ocsp_request.h"	
Description:	Get version of the OCSP request format.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40811]{DRAFT} ⌈

Kind:	function	
Symbol:	Version()	
Scope:	class ara::crypto::x509::OcspResponse	
Syntax:	`virtual std::uint32_t Version () const noexcept=0;`	
Return value:	std::uint32_t	OCSP response format version

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Description:	Get version of the OCSP response format.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40413]{DRAFT} ⌈

Kind:	function	
Symbol:	GetAttribute(AttributeId id)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<ara::core::StringView> GetAttribute (AttributeId id) const noexcept=0;`	
Parameters (in):	id	the identifier of required attribute
Return value:	ara::core::Result< ara::core::StringView >	StringView of the attribute
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknownIdentifier	if the id argument has unsupported value
	SecurityErrorDomain::kInsufficientCapacity	if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Get DN attribute by its ID (this method is applicale to all attributes except kOrgUnit and kDomainComponent). Capacity of the output string must be enough for storing the output value! If (attribute == nullptr) then method only returns required buffer capacity.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40415]{DRAFT} ⌈

Kind:	function	
Symbol:	GetAttribute(AttributeId id, unsigned index)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<ara::core::StringView> 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::StringView >	StringView of the attribute
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
	SecurityErrorDomain::kUnknownIdentifier	if the id argument has unsupported value
	SecurityErrorDomain::kInsufficientCapacity	if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output

▽

△

	SecurityErrorDomain::kInvalid Argument	if (id != kOrgUnit) && (id != kDomainComponent) && (index > 0)
	SecurityErrorDomain::kAbove Boundary	if ((id == kOrgUnit) || (id == kDomainComponent)) and the index value is greater than or equal to the actual number of components in the specified attribute
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Return DN attribute by its ID and sequential index (this method is applicale to attributes kOrg Unit and kDomainComponent). Capacity of the output string must be enough for storing the output value! If (attribute == nullptr) then method only returns required buffer capacity.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40411]{DRAFT} ⌈

Kind:	function	
Symbol:	GetDnString()	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<ara::core::StringView> GetDnString () const noexcept=0;`	
Return value:	ara::core::Result< ara::core::String View >	StringView of the whole DN string
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if (dn != nullptr), but dn->capacity() is less than required for the output value storing
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Get the whole Distinguished Name (DN) as a single string. Capacity of the output string must be enough for storing the output value! If (dn == nullptr) then method only returns required buffer capacity.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40417]{DRAFT} ⌈

Kind:	function	
Symbol:	operator==(const X509DN &other)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual bool 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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Check for equality of this and another Distinguished Name (DN) objects.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40418]{DRAFT} ⌈

Kind:	function	
Symbol:	operator!=(const X509DN &other)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`bool 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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Check for inequality of this and another Distinguished Name (DN) objects.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40414]{DRAFT} ⌈

Kind:	function	
Symbol:	SetAttribute(AttributeId id, ara::core::StringView attribute)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<void> SetAttribute (AttributeId id, ara::core::StringView attribute) const noexcept=0;`	
Parameters (in):	id	the identifier of required attributet
	attribute	the attribute value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if the id argument has unsupported value
	SecurityErrorDomain::kUnexpected Value	if the attribute string contains incorrect characters or it has unsupported length
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Set DN attribute by its ID (this method is applicale to all attributes except kOrgUnit and k DomainComponent).	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40416]{DRAFT} ⌈

Kind:	function	
Symbol:	SetAttribute(AttributeId id, unsigned index, ara::core::StringView attribute)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<void> SetAttribute (AttributeId id, unsigned index, ara::core::StringView attribute) const noexcept=0;`	
	id	the identifier of required attribute
	index	the zero-based index of required component of the attribute

▽

△

	attribute	the attribute value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnknown Identifier	if the id argument has unsupported value
	SecurityErrorDomain::kUnexpected Value	if the attribute string contains incorrect characters or it has unsupported length
	SecurityErrorDomain::kInvalid Argument	if (id != kOrgUnit) && (id != kDomainComponent) && (index > 0)
	SecurityErrorDomain::kAbove Boundary	if ((id == kOrgUnit) || (id == kDomainComponent)) and the index value is greater than the current number of components in the specified attribute
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Set DN attribute by its ID and sequential index (this method is applicable to attributes kOrgUnit and kDomainComponent).	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40412]{DRAFT} ⌈

Kind:	function	
Symbol:	SetDn(ara::core::StringView dn)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<void> SetDn (ara::core::StringView dn) noexcept=0;`	
Parameters (in):	dn	the single string containing the whole DN value in text format
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Set whole Distinguished Name (DN) from a single string. [Error]: SecurityErrorDomain::k UnexpectedValue if the dn string has incorrect syntax.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40511]{DRAFT} ⌈

Kind:	function	
Symbol:	Count()	
Scope:	class ara::crypto::x509::X509Extensions	
Syntax:	`virtual std::size_t Count () const noexcept=0;`	
Return value:	std::size_t	number of elements in the sequence
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

Header file:	#include "ara/crypto/x509/x509_extensions.h"
Description:	Count number of elements in the sequence.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40911]{DRAFT} ⌈

Kind:	function	
Symbol:	MyProvider()	
Scope:	class ara::crypto::x509::X509Object	
Syntax:	`virtual X509Provider& MyProvider () const noexcept=0;`	
Return value:	X509Provider &	a reference to X.509 Provider instance that provides this object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_object.h"	
Description:	Get a reference to X.509 Provider of this object.	

⌋(*RS_CRYPTO_02401*)

[SWS_CRYPT_40612]{DRAFT} ⌈

Kind:	function	
Symbol:	BuildDn(ara::core::StringView dn)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<X509DN::Uptrc> BuildDn (ara::core::StringView 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if the dn argument has incorrect format
	SecurityErrorDomain::kInvalidInputSize	if the dn argument has unsupported length (too large)
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create completed X.500 Distinguished Name structure from the provided string representation.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40629]{DRAFT} ⌈

Kind:	function	
Symbol:	CheckCertStatus(Certificate &cert, const OcspResponse &ocspResponse)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<bool> CheckCertStatus (Certificate &cert, const OcspResponse &ocspResponse) const noexcept=0;`	
Parameters (in):	cert	a certificate that should be verified
	ocspResponse	an OCSP response
Return value:	ara::core::Result< bool >	true if the certificate is verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the cert is invalid
	SecurityErrorDomain::kRuntimeFault	if the ocspResponse is invalid
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Check certificate status by directly provided OCSP response. This method may be used for implementation of the "OCSP stapling". This method updates the Certificate::Status associated with the certificate.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40630]{DRAFT} ⌈

Kind:	function	
Symbol:	CheckCertStatus(const ara::core::Vector< Certificate * > &certList, const OcspResponse &ocsp Response)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<bool> CheckCertStatus (const ara::core::Vector< Certificate * > &certList, const OcspResponse &ocsp Response) const noexcept=0;`	
Parameters (in):	certList	a certificates list that should be verified
	ocspResponse	an OCSP response
Return value:	ara::core::Result< bool >	true if the certificates list is verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the provided certificates are invalid
	SecurityErrorDomain::kRuntimeFault	if the ocspResponse is invalid
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Check status of a certificates list by directly provided OCSP response. This method may be used for implementation of the "OCSP stapling". This method updates the Certificate::Status associated with the certificates in the list.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40635]{DRAFT} ⌈

Kind:	function
Symbol:	CleanupVolatileStorage()
Scope:	class ara::crypto::x509::X509Provider
Syntax:	`virtual void CleanupVolatileStorage () noexcept=0;`
Return value:	None
Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	Cleanup the volatile certificates storage. After execution of this command the certificates previously imported to the volatile storage cannot be found by a search, but it doesn't influence to already loaded Certificate instances! .

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40640]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateCertSignRequest(cryp::SignerPrivateCtx::Uptr signerCtx, ReadOnlyMemRegion derSubjectDN, ReadOnlyMemRegion x509Extensions=ReadOnlyMemRegion(), unsigned version=1)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<CertSignRequest::Uptrc> CreateCertSignRequest (cryp::SignerPrivateCtx::Uptr signerCtx, ReadOnlyMemRegion derSubjectDN, ReadOnlyMemRegion x509Extensions=ReadOnlyMemRegion(), unsigned version=1) const noexcept=0;`	
Parameters (in):	signerCtx	the fully-configured SignerPrivateCtx to be used for signing this certificate request
	derSubjectDN	the DER-encoded subject distinguished name (DN) of the private key owner
	x509Extensions	the DER-encoded X.509 Extensions that should be included to the certification request
	version	the format version of the target certification request
Return value:	ara::core::Result< CertSignRequest::Uptrc >	unique smart pointer to created certification request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnexpectedValue	if any of arguments has incorrect/unsupported value
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create certification request for a private key loaded to the context.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40615]{DRAFT} ⌈

Kind:	function	
Symbol:	CountCertsInChain(ReadOnlyMemRegion certChain, Serializable::FormatId formatId=Serializable::kFormatDefault)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<std::size_t> CountCertsInChain (ReadOnlyMemRegion certChain, Serializable::FormatId formatId=Serializable::kFormatDefault) const noexcept=0;`	
Parameters (in):	certChain	DER/PEM-encoded certificate chain (in form of a single BLOB)
	formatId	input format identifier (kFormatDefault means auto-detect)
Return value:	ara::core::Result< std::size_t >	number of certificates in the chain
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalidArgument	if the certChain argument cannot be pre-parsed
	SecurityErrorDomain::kUnknownIdentifier	if the formatId argument has unknown value
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Count number of certificates in a serialized certificate chain represented by a single BLOB.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40611]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateEmptyDn(std::size_t capacity=0)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<X509DN::Uptr> CreateEmptyDn (std::size_t capacity=0) noexcept=0;`	
Parameters (in):	capacity	number of bytes that should be reserved for the content of the target X509DN object
Return value:	ara::core::Result< X509DN::Uptr >	Unique smart pointer to created empty X509DN object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create an empty X.500 Distinguished Name (DN) structure. If (0 == capacity) then a maximally supported (by the implementation) capacity must be reserved.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40636]{DRAFT} ⌈

Kind:	function
Symbol:	CreateEmptyExtensions(std::size_t capacity=0)
Scope:	class ara::crypto::x509::X509Provider

▽

△

Syntax:	`virtual ara::core::Result<X509Extensions::Uptr> CreateEmptyExtensions (std::size_t capacity=0) noexcept=0;`	
Parameters (in):	capacity	number of bytes that should be reserved for the content of the target X509Extensions object
Return value:	ara::core::Result< X509Extensions::Uptr >	Shared smart pointer to created empty X509X509Extensions object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create an empty X.509 Extensions structure. If (0 == capacity) then a maximally supported (by the implementation) capacity must be reserved.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40626]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateOcspRequest(const Certificate &cert, ara::core::Optional< const cryp::SignerPrivateCtx > signer)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<OcspRequest::Uptrc> CreateOcspRequest (const Certificate &cert, ara::core::Optional< const cryp::SignerPrivateCtx > signer) noexcept=0;`	
Parameters (in):	cert	a certificate that should be verified
	signer	an optional pointer to initialized signer context (if the request should be signed)
Return value:	ara::core::Result< OcspRequest::Uptrc >	unique smart pointer to the created OCSP request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the provided certificate is invalid
	SecurityErrorDomain::kIncompleteArg State	if the signer context is not initialized by a key
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create OCSP request for specified certificate. This method may be used for implementation of the "OCSP stapling".	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40627]{DRAFT} ⌈

Kind:	function
Symbol:	CreateOcspRequest(const ara::core::Vector< const Certificate * > &certList, ara::core::Optional< const cryp::SignerPrivateCtx > signer)
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 > signer) noexcept=0;`	
Parameters (in):	certList	a certificates' list that should be verified
	signer	an optional pointer to initialized signer context (if the request should be signed)
Return value:	ara::core::Result< OcspRequest::Uptrc >	unique smart pointer to the created OCSP request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the provided certificates are invalid
	SecurityErrorDomain::kIncompleteArg State	if the signer context is not initialized by a key
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create OCSP request for specified list of certificates. This method may be used for implementation of the "OCSP stapling".	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40613]{DRAFT} ⌈

Kind:	function	
Symbol:	DecodeDn(ReadOnlyMemRegion dn, Serializable::FormatId formatId=Serializable::kFormat Default)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<X509DN::Uptrc> DecodeDn (ReadOnlyMemRegion dn, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0;`	
Parameters (in):	dn	DER/PEM-encoded representation of the Distinguished Name
	formatId	input format identifier (kFormatDefault means auto-detect)
Return value:	ara::core::Result< X509DN::Uptrc >	unique smart pointer for the created X509DN object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the dn argument cannot be parsed
	SecurityErrorDomain::kUnknown Identifier	if the formatId argument has unknown value
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Decode X.500 Distinguished Name structure from the provided serialized format.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40631]{DRAFT} ⌈

Kind:	function	
Symbol:	FindCertByDn(const X509DN &subjectDn, const X509DN &issuerDn, time_t validityTimePoint, StorageIndex &certIndex)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual Certificate::Uptrc FindCertByDn (const X509DN &subjectDn, const X509DN &issuerDn, time_t validityTimePoint, StorageIndex &certIndex) 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
Parameters (inout):	certIndex	an index for iteration through all suitable certificates in the storage (input: index of previous found cerificate, output: index of current found cerificate)
Return value:	Certificate::Uptrc	unique smart pointer to found certificate or nullptr if nothing is found
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Find a certificate by the subject and issuer Distinguished Names (DN). Argument certIndex represents an internal index of current certificate in the storage. In order to start certificate search from begin, set: certIndex = kInvalidIndex	

⌋ *(RS_CRYPTO_02306)*

[SWS_CRYPT_40632]{DRAFT} ⌈

Kind:	function	
Symbol:	FindCertByKeyIds(ReadOnlyMemRegion subjectKeyId, ReadOnlyMemRegion authorityKeyId=ReadOnlyMemRegion())	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual Certificate::Uptrc FindCertByKeyIds (ReadOnlyMemRegion subjectKeyId, ReadOnlyMemRegion authorityKeyId=ReadOnlyMemRegion()) noexcept=0;`	
Parameters (in):	subjectKeyId	subject key identifier (SKID)
	authorityKeyId	optional authority key identifier (AKID)
Return value:	Certificate::Uptrc	unique smart pointer to found certificate or nullptr if nothing is found
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Find a certificate by its SKID & AKID.	

⌋ *(RS_CRYPTO_02306)*

[SWS_CRYPT_40633]{DRAFT} ⌈

Kind:	function	
Symbol:	FindCertBySn(ReadOnlyMemRegion sn, const X509DN &issuerDn)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual 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:	Certificate::Uptrc	unique smart pointer to a found certificate or nullptr if nothing is found
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Find a certificate by its serial number.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40634]{DRAFT} ⌈

Kind:	function	
Symbol:	ParseCertSignRequest(ReadOnlyMemRegion csr, bool withMetaData=true)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<CertSignRequest::Uptrc> ParseCertSignRequest (ReadOnlyMemRegion csr, bool withMetaData=true) noexcept=0;`	
Parameters (in):	csr	the buffer containing a certificate signing request
	withMetaData	specifies the format of the buffer content: TRUE means the object has been previously serialized by using the Serializable interface; FALSE means the CSR was exported using the CertSign Request::ExportASN1CertSignRequest() interface
Return value:	ara::core::Result< CertSign Request::Uptrc >	unique smart pointer to the certificate signing request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnsupported Format	is returned in case the provided buffer does not contain the expected format
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse a certificate signing request (CSR) provided by the user.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40620]{DRAFT} ⌈

Kind:	function
Symbol:	ImportCrl(ReadOnlyMemRegion crl)
Scope:	class ara::crypto::x509::X509Provider
Syntax:	`virtual ara::core::Result<bool> ImportCrl (ReadOnlyMemRegion crl) noexcept=0;`

▽

△

Parameters (in):	crl	serialized CRL or Delta CRL (in form of a BLOB)
Return value:	ara::core::Result< bool >	true if the CRL is valid and false if it is already expired
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnexpected Value	if the provided BLOB is not a CRL/DeltaCRL
	SecurityErrorDomain::kRuntimeFault	if the CRL validation has failed
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Import Certificate Revocation List (CRL) or Delta CRL from a memory BLOB. If the imported CRL lists some certificates kept in the persistent or volatile storages then their status must be automatically updated to Status::kInvalid. If some of these certificates were already openned during this operation, then this status change becomes available immediately (via method call Certificate::GetStatus())! All certificates with the status kInvalid should be automatically removed from correspondent storages (immediately if a certificate not in use now or just after its closing otherwise).	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40621]{DRAFT} ⌈

Kind:	function	
Symbol:	Import(const Certificate &cert, ara::core::Optional< ara::core::InstanceSpecifier > iSpecify)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<void> Import (const Certificate &cert, ara::core::Optional< ara::core::InstanceSpecifier > iSpecify) noexcept=0;`	
Parameters (in):	cert	a valid certificate that should be imported
	iSpecify	optionally a valid InstanceSpecifier can be provided that points to a CertificateSlot for persistent storage of the certificate, otherwise the certificate shall be stored in volatile (session) storage
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the provided certificate is invalid
	SecurityErrorDomain::kIncompatible Object	if provided certificate has partial collision with a matched CSR in the storage
	SecurityErrorDomain::kContent Duplication	if the provided certificate already exists in the storage
	SecurityErrorDomain::kAccess Violation	if the InstanceSpecifier points to a CertificateSlot, which the application may only read
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Import the certificate to volatile or persistent storage. Only imported certificate may be found by a search and applied for automatic verifications! A certificate can be imported to only one of storage: volatile or persistent. Therefore if you import a certificate already kept in the persistent storage to the volatile one then nothing changes. But if you import a certificate already kept in the volatile storage to the persistent one then it is "moved" to the persistent realm. If an application successfully imports a certificate that correspond to a CSR existing in the storage then this CSR should be removed.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40641]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadCertificate(ara::core::InstanceSpecifier &iSpecify)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<Certificate::Uptr> 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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnreserved Resource	if the InstanceSpecifier is incorrect (the certificate cannot be found)
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Load a certificate from the persistent certificate storage.	

⌋*()*

[SWS_CRYPT_40616]{DRAFT} ⌈

Kind:	function	
Symbol:	ParseCertChain(ara::core::Vector< Certificate::Uptr > &outcome, ReadOnlyMemRegion cert Chain, Serializable::FormatId formatId=Serializable::kFormatDefault)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<void> ParseCertChain (ara::core::Vector< Certificate::Uptr > &outcome, ReadOnlyMemRegion certChain, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0;`	
Parameters (in):	certChain	DER/PEM-encoded certificate chain (in form of a single BLOB)
	formatId	input format identifier (kFormatDefault means auto-detect)
Parameters (out):	outcome	an output vector for imported certificates
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficient Capacity	if the capacity of outcome vector is less than actual number of certificates in the chain
	SecurityErrorDomain::kInvalid Argument	if the certChain argument cannot be parsed
	SecurityErrorDomain::kUnknown Identifier	if the formatId argument has unknown value
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse a serialized representation of the certificate chain and create their instances. Off-line validation of the parsed certification chain may be done via call VerifyCertChainByCrl(). After validation the certificates may be saved to the session or persistent storage for following search and usage. If the certificates are not imported then they will be lost after destroy of the returned instances! Only imported certificates may be found by a search and applied for automatic verifications! Certificates in the outcome vector will be placed from the root CA certificate (zero index) to the final end-entity certificate (last used index of the vector).	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40617]{DRAFT} ⌈

Kind:	function	
Symbol:	ParseCertChain(ara::core::Vector< Certificate::Uptr > &outcome, const ara::core::Vector< ReadOnlyMemRegion > &certChain, Serializable::FormatId formatId=Serializable::kFormatDefault)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<void> ParseCertChain (ara::core::Vector< Certificate::Uptr > &outcome, const ara::core::Vector< ReadOnlyMemRegion > &certChain, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0;`	
Parameters (in):	certChain	DER/PEM-encoded certificates chain (each certificate is presented by a separate BLOB in the input vector)
	formatId	input format identifier (kFormatDefault means auto-detect)
Parameters (out):	outcome	output vector of imported certificates
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficientCapacity	if capacity of the outcome vector is less than number of elements in the certChain
	SecurityErrorDomain::kInvalidArgument	if an element of certChain argument cannot be parsed
	SecurityErrorDomain::kUnknownIdentifier	if the formatId argument has unknown value
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse a serialized representation of the certificate chain and create their instances. Off-line validation of the parsed certification chain may be done via call VerifyCertChainByCrl(). After validation the certificates may be imported to the session or persistent storage for following search and usage. Capacity of the outcome vector must be equal to the size of the certChain vector. If the certificates are not imported then they will be lost after destroy of the returned instances! Only imported certificates may be found by a search and applied for automatic verifications! Certificates in the outcome vector will be placed from the root CA certificate (zero index) to the final end-entity certificate (last used index of the vector).	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40614]{DRAFT} ⌈

Kind:	function	
Symbol:	ParseCert(ReadOnlyMemRegion cert, Serializable::FormatId formatId=Serializable::kFormatDefault)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<Certificate::Uptr> ParseCert (ReadOnlyMemRegion cert, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0;`	
Parameters (in):	cert	DER/PEM-encoded certificate
	formatId	input format identifier (kFormatDefault means auto-detect)
Return value:	ara::core::Result< Certificate::Uptr >	unique smart pointer to created certificate
Exception Safety:	noexcept	

▽

△

Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the cert argument cannot be parsed
	SecurityErrorDomain::kUnknown Identifier	if the formatId argument has unknown value
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse a serialized representation of the certificate and create its instance. Off-line validation of the parsed certificate may be done via call VerifyCertByCrl(). After validation the certificate may be imported to the session or persistent storage for following search and usage. If the parsed certificate is not imported then it will be lost after destroy of the returned instance! Only imported certificate may be found by a search and applied for automatic verifications!	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40628]{DRAFT} ⌈

Kind:	function	
Symbol:	ParseOcspResponse(ReadOnlyMemRegion response)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual OcspResponse::Uptrc ParseOcspResponse (ReadOnlyMemRegion response) const noexcept=0;`	
Parameters (in):	response	a serialized OCSP response
Return value:	OcspResponse::Uptrc	unique smart pointer to the created OCSP response instance
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnexpected Value	if the provided BLOB response doesn't keep an OCSP response
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse serialized OCSP response and create correspondent interface instance. This method may be used for implementation of the "OCSP stapling".	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40622]{DRAFT} ⌈

Kind:	function	
Symbol:	Remove(Certificate::Uptrc &&cert)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual bool Remove (Certificate::Uptrc &&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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	

▽

△

Description:	Remove specified certificate from the storage (volatile or persistent) and destroy it.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40625]{DRAFT} ⌈

Kind:	function	
Symbol:	SetAsRootOfTrust(const Certificate &caCert)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<void> SetAsRootOfTrust (const Certificate &caCert) noexcept=0;`	
Parameters (in):	caCert	a valid CA certificate that should be trusted
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInvalid Argument	if the provided certificate is invalid
	SecurityErrorDomain::kIncompatible Object	if provided certificate doesn't belong to a CA
	SecurityErrorDomain::kAccess Violation	if the method called by an application without the "Trust Master" permission
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Set specified CA certificate as a "root of trust". Only a certificate saved to the volatile or persistent storage may be marked as the "root of trust"! Only CA certificate can be a "root of trust"! Multiple certificates on an ECU may be marked as the "root of trust". Only an application with permissions "Trust Master" has the right to call this method!	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40624]{DRAFT} ⌈

Kind:	function	
Symbol:	SetPendingStatus(const CertSignRequest &request)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<void> SetPendingStatus (const CertSignRequest &request) noexcept=0;`	
Parameters (in):	request	certificate signing request that should be marked as "pending"
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kAccess Violation	if the method called by an application without the "CA Connector" permission
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Set the "pending" status associated to the CSR that means that the CSR already sent to CA. This method do nothing if the CSR already marked as "pending". Only an application with permissions "CA Connector" has the right to call this method!	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40618]{DRAFT} ⌈

Kind:	function	
Symbol:	VerifyCert(Certificate &cert, Certificate::Uptr myRoot=nullptr)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual Certificate::Status VerifyCert (Certificate &cert, Certificate::Uptr myRoot=nullptr) noexcept=0;`	
Parameters (in):	cert	target certificate for verification
	myRoot	root certificate to be used for verification - if this is nullptr, use machine root certificates
Return value:	Certificate::Status	verification status of the provided certificate
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Verify status of the provided certificate by locally stored CA certificates and CRLs only. This method updates the Certificate::Status associated with the certificate.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40619]{DRAFT} ⌈

Kind:	function	
Symbol:	VerifyCertChain(ara::core::Span< const Certificate::Uptr > chain, Certificate::Uptr my Root=nullptr)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual Certificate::Status VerifyCertChain (ara::core::Span< const Certificate::Uptr > chain, Certificate::Uptr myRoot=nullptr) const noexcept=0;`	
Parameters (in):	chain	target certificate chain for verification
	myRoot	root certificate to be used for verification - if this is nullptr, use machine root certificates
Return value:	Certificate::Status	verification status of the provided certificate chain
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Verify status of the provided certification chain by locally stored CA certificates and CRLs only. Verification status of the certificate chain is Certificate::Status::kValid only if all certificates in the chain have such status! Certificates in the chain (presented by container vector) must be placed from the root CA certificate (zero index) to the target end-entity certificate (last used index of the vector). Verification is executed in same order. If the chain verification is failed then status of the first failed certificate is returned. This method updates the Certificate::Status associated with the certificates in the chain.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40604]{DRAFT} ⌈

Kind:	function
Symbol:	~X509Provider()
Scope:	class ara::crypto::x509::X509Provider
Syntax:	`virtual ~X509Provider () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	Destructor.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_30226]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const X509Provider &other)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`X509Provider& operator= (const X509Provider &other)=default;`	
Parameters (in):	other	the other instance
Return value:	X509Provider &	*this, containing the contents of other
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Copy-assign another X509Provider to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30227]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(X509Provider &&other)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`X509Provider& operator= (X509Provider &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	X509Provider &	*this, containing the contents of other
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Move-assign another X509Provider to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_40101]{DRAFT} ⌈

Kind:	type alias
Symbol:	KeyConstraints
Scope:	class ara::crypto::x509::BasicCertInfo
Derived from:	std::uint32_t
Syntax:	`using KeyConstraints = std::uint32_t;`

▽

△

Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	X.509 v3 Key Constraints type definition.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40203]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	Status	
Scope:	class ara::crypto::x509::Certificate	
Underlying type:	std::uint32_t	
Syntax:	`enum class Status : std::uint32_t {...};`	
Values:	kValid= 0	The certificate is valid.
	kInvalid= 1	The certificate is invalid.
	kUnknown= 2	Status of the certificate is unknown yet.
	kNoTrust= 3	The certificate has correct signature, but the ECU has no a root of trust for this certificate.
	kExpired= 4	The certificate has correct signature, but it is already expired (its validity period has ended)
	kFuture= 5	The certificate has correct signature, but its validity period is not started yet.
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Certificate verification status.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40202]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::x509::Certificate
Derived from:	std::unique_ptr<const Certificate>
Syntax:	`using Uptrc = std::unique_ptr<const Certificate>;`
Header file:	#include "ara/crypto/x509/certificate.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40201]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::x509::Certificate
Derived from:	std::unique_ptr<Certificate>

▽

△

Syntax:	`using Uptr = std::unique_ptr<Certificate>;`
Header file:	#include "ara/crypto/x509/certificate.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40301]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::x509::CertSignRequest
Derived from:	std::unique_ptr<const CertSignRequest>
Syntax:	`using Uptrc = std::unique_ptr<const CertSignRequest>;`
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Description:	Unique smart pointer of the constant interface.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40302]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::x509::CertSignRequest
Derived from:	std::unique_ptr<CertSignRequest>
Syntax:	`using Uptr = std::unique_ptr<CertSignRequest>;`
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40002]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	OcspCertStatus	
Scope:	namespace ara::crypto::x509	
Underlying type:	std::uint32_t	
Syntax:	`enum class OcspCertStatus : std::uint32_t {...};`	
Values:	kGood= 0	The certificate is not revoked.
	kRevoked= 1	The certificate has been revoked (either permanantly or temporarily (on hold))
	kUnknown= 2	The responder doesn't know about the certificate being requested.
Header file:	#include "ara/crypto/x509/ocsp_response.h"	

▽

△

Description:	On-line Certificate Status Protocol (OCSP) Certificate Status.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40702]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::x509::OcspRequest
Derived from:	std::unique_ptr<const OcspRequest>
Syntax:	`using Uptrc = std::unique_ptr<const OcspRequest>;`
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40701]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::x509::OcspRequest
Derived from:	std::unique_ptr<OcspRequest>
Syntax:	`using Uptr = std::unique_ptr<OcspRequest>;`
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40001]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	OcspResponseStatus	
Scope:	namespace ara::crypto::x509	
Underlying type:	std::uint32_t	
Syntax:	`enum class OcspResponseStatus : std::uint32_t {...};`	
Values:	kSuccessful= 0	Response has valid confirmations.
	kMalformedRequest= 1	Illegal confirmation request.
	kInternalError= 2	Internal error in issuer.
	kTryLater= 3	Try again later.
	kSigRequired= 5	Must sign the request.
	kUnauthorized= 6	Request unauthorized.
Header file:	#include "ara/crypto/x509/ocsp_response.h"	
Description:	On-line Certificate Status Protocol (OCSP) Response Status.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40802]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::x509::OcspResponse
Derived from:	std::unique_ptr<const OcspResponse>
Syntax:	`using Uptrc = std::unique_ptr<const OcspResponse>;`
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40801]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::x509::OcspResponse
Derived from:	std::unique_ptr<OcspResponse>
Syntax:	`using Uptr = std::unique_ptr<OcspResponse>;`
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40403]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	AttributeId	
Scope:	class ara::crypto::x509::X509DN	
Underlying type:	std::uint32_t	
Syntax:	`enum class AttributeId : std::uint32_t {...};`	
	kCommonName= 0	Common Name.
	kCountry= 1	Country.
	kState= 2	State.
	kLocality= 3	Locality.
	kOrganization= 4	Organization.
	kOrgUnit= 5	Organization Unit.
	kStreet= 6	Street.
	kPostalCode= 7	Postal Code.
	kTitle= 8	Title.
	kSurname= 9	Surname.
	kGivenName= 10	Given Name.
	kInitials= 11	Initials.
	kPseudonym= 12	Pseudonym.
	kGenerationQualifier= 13	Generation Qualifier.

▽

△

	kDomainComponent= 14	Domain Component.
	kDnQualifier= 15	Distinguished Name Qualifier.
	kEmail= 16	E-mail.
	kUri= 17	URI.
	kDns= 18	DNS.
	kHostName= 19	Host Name (UNSTRUCTUREDNAME)
	kIpAddress= 20	IP Address (UNSTRUCTUREDADDRESS)
	kSerialNumbers= 21	Serial Numbers.
	kUserId= 22	User ID.
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Enumeration of DN attributes' identifiers.	

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40402]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::x509::X509DN
Derived from:	std::unique_ptr<const X509DN>
Syntax:	using Uptrc = std::unique_ptr<const X509DN>;
Header file:	#include "ara/crypto/x509/x509_dn.h"
Description:	Unique smart pointer of the constant interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40401]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::x509::X509DN
Derived from:	std::unique_ptr<X509DN>
Syntax:	using Uptr = std::unique_ptr<X509DN>;
Header file:	#include "ara/crypto/x509/x509_dn.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40501]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::x509::X509Extensions

▽

△

Derived from:	std::unique_ptr<X509Extensions>
Syntax:	`using Uptr = std::unique_ptr<X509Extensions>;`
Header file:	#include "ara/crypto/x509/x509_extensions.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_24401]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::x509::X509PublicKeyInfo
Derived from:	std::unique_ptr<const X509PublicKeyInfo>
Syntax:	`using Uptrc = std::unique_ptr<const X509PublicKeyInfo>;`
Header file:	#include "ara/crypto/x509/x509_public_key_info.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02307)*

[SWS_CRYPT_40601]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::x509::X509Provider
Derived from:	std::unique_ptr<X509Provider>
Syntax:	`using Uptr = std::unique_ptr<X509Provider>;`
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40602]{DRAFT} ⌈

Kind:	type alias
Symbol:	StorageIndex
Scope:	class ara::crypto::x509::X509Provider
Derived from:	std::size_t
Syntax:	`using StorageIndex = std::size_t;`
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	Type of an internal index inside the certificate storage.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40157]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrCrlSign
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrCrlSign = 0x0200;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for Certificates Revokation Lists (CRL) signing.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40154]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrDataEncipherment
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrDataEncipherment = 0x1000;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for data encipherment.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40159]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrDecipherOnly
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrDecipherOnly = 0x0080;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The enciphermet key can be used for deciphering only.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40151]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrDigitalSignature
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrDigitalSignature = 0x8000;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for digital signature production.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40158]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrEncipherOnly
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrEncipherOnly = 0x0100;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The enciphermet key can be used for enciphering only.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40155]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrKeyAgreement
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrKeyAgreement = 0x0800;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for a key agreement protocol execution.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40156]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrKeyCertSign
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrKeyCertSign = 0x0400;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for certificates signing.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40153]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrKeyEncipherment
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrKeyEncipherment = 0x2000;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for key encipherment.

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_40152]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrNonRepudiation
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrNonRepudiation = 0x4000;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used in cases requiring the "non-repudiation" guarantee.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40150]{DRAFT} ⌈

Kind:	variable
Symbol:	kConstrNone
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints kConstrNone = 0;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	No key constraints.

⌋*(RS_CRYPTO_02306)*

[SWS_CRYPT_40603]{DRAFT} ⌈

Kind:	variable
Symbol:	kInvalidIndex
Scope:	class ara::crypto::x509::X509Provider
Type:	const StorageIndex
Syntax:	`const StorageIndex kInvalidIndex = static_cast<std::size_t>(-1LL);`
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	Reserved "invalid index" value for navigation inside the certificate storage.

⌋*(RS_CRYPTO_02306)*

8.4 API Common Data Types

[SWS_CRYPT_10015]{DRAFT} ⌈

Kind:	type alias
Symbol:	AllowedUsageFlags
Scope:	namespace ara::crypto

▽

△

Derived from:	std::uint32_t
Syntax:	`using AllowedUsageFlags = std::uint32_t;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A container type and constant bit-flags of allowed usages of a key or a secret seed object. Only directly specified usages of a key are allowed, all other are prohibited! Similar set of flags are defined for the usage restrictions of original key/seed and for a symmetric key or seed that potentially can be derived from the original one. A symmetric key or secret seed can be derived from the original one, only if it supports kAllowKeyAgreement or kAllowKeyDiversify or kAllow KeyDerivation!

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_10042]{DRAFT} ⌈

Kind:	type alias	
Symbol:	ByteVector	
Scope:	namespace ara::crypto	
Derived from:	ara::core::Vector<std::uint8_t, Alloc>	
Syntax:	`using ByteVector = ara::core::Vector<std::uint8_t, Alloc>;`	
Template param:	Alloc	custom allocator of bytes sequences
Header file:	#include "ara/crypto/common/base_id_types.h"	
Description:	Alias of a bytes' vector template with customizable allocator.	

⌋*(RS_CRYPTO_02201, RS_CRYPTO_02202, RS_CRYPTO_02203, RS_CRYPTO_02204, RS_CRYPTO_02205, RS_CRYPTO_02206, RS_CRYPTO_02207, RS_CRYPTO_02208, RS_CRYPTO_02209)*

[SWS_CRYPT_10014]{DRAFT} ⌈

Kind:	type alias
Symbol:	CryptoAlgId
Scope:	namespace ara::crypto
Derived from:	std::uint64_t
Syntax:	`using CryptoAlgId = std::uint64_t;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Container type of the Crypto Algorithm Identifier.

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107)*

[SWS_CRYPT_10016]{DRAFT} ⌈

Kind:	enumeration
Symbol:	CryptoObjectType
Scope:	namespace ara::crypto

▽

△

Underlying type:	std::uint32_t	
Syntax:	`enum class CryptoObjectType : std::uint32_t {...};`	
Values:	kUndefined= 0	Object type is currently not defined (empty container)
	kSymmetricKey= 1	cryp::SymmetricKey object
	kPrivateKey= 2	cryp::PrivateKey object
	kPublicKey= 3	cryp::PublicKey object
	kSignature= 4	cryp::Signature object (asymmetric digital signature or symmetric MAC/HMAC or hash digest)
	kSecretSeed= 5	cryp::SecretSeed object. Note: the seed cannot have an associated crypto algorithm!
Header file:	#include "ara/crypto/common/base_id_types.h"	
Description:	Enumeration of all types of crypto objects, i.e. types of content that can be stored to a key slot.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_10100]{DRAFT} ⌈

Kind:	struct
Symbol:	CryptoObjectUid
Scope:	namespace ara::crypto
Syntax:	`struct CryptoObjectUid {...};`
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Description:	Definition of Crypto Object Unique Identifier (COUID) type.

⌋*(RS_CRYPTO_02005, RS_CRYPTO_02006)*

[SWS_CRYPT_10017]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ProviderType	
Scope:	namespace ara::crypto	
Underlying type:	std::uint32_t	
Syntax:	`enum class ProviderType : std::uint32_t {...};`	
Values:	kUndefinedProvider= 0	Undefined/Unknown Provider type (or applicable for the whole Crypto Stack)
	kCryptoProvider= 1	Cryptography Provider.
	kKeyStorageProvider= 2	Key Storage Provider.
	kX509Provider= 3	X.509 Provider.
Header file:	#include "ara/crypto/common/base_id_types.h"	
Description:	Enumeration of all known Provider types.	

⌋*(RS_CRYPTO_02401, RS_CRYPTO_02109)*

[SWS_CRYPT_10033]{DRAFT} ⌈

Kind:	type alias
Symbol:	ReadOnlyMemRegion
Scope:	namespace ara::crypto
Derived from:	ara::core::Span<const std::uint8_t>
Syntax:	`using ReadOnlyMemRegion = ara::core::Span<const std::uint8_t>;`
Header file:	#include "ara/crypto/common/mem_region.h"
Description:	Read-Only Memory Region (intended for [in] arguments)

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10031]{DRAFT} ⌈

Kind:	type alias
Symbol:	ReadWriteMemRegion
Scope:	namespace ara::crypto
Derived from:	ara::core::Span<std::uint8_t>
Syntax:	`using ReadWriteMemRegion = ara::core::Span<std::uint8_t>;`
Header file:	#include "ara/crypto/common/mem_region.h"
Description:	Read-Write Memory Region (intended for [in/out] arguments)

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10099]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	SecurityErrc	
Scope:	namespace ara::crypto	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class SecurityErrc : ara::core::ErrorDomain::CodeType {...};`	
	kErrorClass= 0x1000000	Reserved (a multiplier of error class IDs)
	kErrorSubClass= 0x10000	Reserved (a multiplier of error sub-class IDs)
	kErrorSubSubClass= 0x100	Reserved (a multiplier of error sub-sub-class IDs)
	kResourceFault= 1 * kErrorClass	ResourceException: Generic resource fault!
	kBusyResource= kResourceFault + 1	ResourceException: Specified resource is busy!
	kInsufficientResource= kResourceFault + 2	ResourceException: Insufficient capacity of specified resource!
	kUnreservedResource= kResource Fault + 3	ResourceException: Specified resource was not reserved!
	kModifiedResource= kResourceFault + 4	ResourceException: Specified resource has been modified!
	kLogicFault= 2 * kErrorClass	LogicException: Generic logic fault!
	kInvalidArgument= kLogicFault + 1 * k ErrorSubClass	InvalidArgumentException: An invalid argument value is provided!
	kUnknownIdentifier= kInvalidArgument + 1	InvalidArgumentException: Unknown identifier is provided!

▽

△

	kInsufficientCapacity= kInvalid Argument + 2	InvalidArgumentException: Insufficient capacity of the output buffer!
	kInvalidInputSize= kInvalidArgument + 3	InvalidArgumentException: Invalid size of an input buffer!
	kIncompatibleArguments= kInvalid Argument + 4	InvalidArgumentException: Provided values of arguments are incompatible!
	kInOutBuffersIntersect= kInvalid Argument + 5	InvalidArgumentException: Input and output buffers are intersect!
	kBelowBoundary= kInvalidArgument + 6	InvalidArgumentException: Provided value is below the lower boundary!
	kAboveBoundary= kInvalidArgument + 7	InvalidArgumentException: Provided value is above the upper boundary!
	kAuthTagNotValid= kInvalidArgument + 8	AuthTagNotValidException: Provided authentication-tag cannot be verified!
	kUnsupported= kInvalidArgument + 1 * kErrorSubSubClass	UnsupportedException: Unsupported request (due to limitations of the implementation)!
	kInvalidUsageOrder= kLogicFault + 2 * kErrorSubClass	InvalidUsageOrderException: Invalid usage order of the interface!
	kUninitializedContext= kInvalidUsage Order + 1	InvalidUsageOrderException: Context of the interface was not initialized!
	kProcessingNotStarted= kInvalidUsage Order + 2	InvalidUsageOrderException: Data processing was not started yet!
	kProcessingNotFinished= kInvalid UsageOrder + 3	InvalidUsageOrderException: Data processing was not finished yet!
	kRuntimeFault= 3 * kErrorClass	RuntimeException: Generic runtime fault!
	kUnsupportedFormat= kRuntimeFault + 1	RuntimeException: Unsupported serialization format for this object type!
	kBruteForceRisk= kRuntimeFault + 2	RuntimeException: Operation is prohibitted due to a risk of a brute force attack!
	kContentRestrictions= kRuntimeFault + 3	RuntimeException: The operation violates content restrictions of the target container!
	kBadObjectReference= kRuntimeFault + 4	RuntimeException: Incorrect reference between objects!
	kContentDuplication= kRuntimeFault + 6	RuntimeException: Provided content already exists in the target storage!
	kUnexpectedValue= kRuntimeFault + 1 * kErrorSubClass	UnexpectedValueException: Unexpected value of an argument is provided!
	kIncompatibleObject= kUnexpected Value + 1	UnexpectedValueException: The provided object is incompatible with requested operation or its configuration!
	kIncompleteArgState= kUnexpected Value + 2	UnexpectedValueException: Incomplete state of an argument!
	kEmptyContainer= kUnexpectedValue + 3	UnexpectedValueException: Specified container is empty!
	kMissingArgument= kUnexpectedValue + 4	kMissingArgumentException: Expected argument, but none provided!
	kBadObjectType= kUnexpectedValue + 1 * kErrorSubSubClass	BadObjectTypeException: Provided object has unexpected type!
	kUsageViolation= kRuntimeFault + 2 * kErrorSubClass	UsageViolationException: Violation of allowed usage for the object!

▽

△

	kAccessViolation= kRuntimeFault + 3 * kErrorSubClass	AccessViolationException: Access rights violation!
Header file:	#include "ara/crypto/common/security_error_domain.h"	
Description:	Enumeration of all Security Error Code values that may be reported by ara::crypto.	

⌋*(RS_CRYPTO_02310)*

[SWS_CRYPT_30001]{DRAFT} ⌈

Kind:	struct
Symbol:	SecureCounter
Scope:	namespace ara::crypto
Syntax:	`struct SecureCounter {...};`
Header file:	#include "ara/crypto/common/entry_point.h"
Description:	128 bit secure counter made up of most significant and least significant quad-word of the hardware counter.

⌋*(RS_CRYPTO_02401)*

[SWS_CRYPT_10701]{DRAFT} ⌈

Kind:	type alias
Symbol:	FormatId
Scope:	class ara::crypto::Serializable
Derived from:	std::uint32_t
Syntax:	`using FormatId = std::uint32_t;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	A container type for the encoding format identifiers.

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02302)*

[SWS_CRYPT_10019]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	CryptoTransform	
Scope:	namespace ara::crypto	
Underlying type:	std::uint32_t	
Syntax:	`enum class CryptoTransform : std::uint32_t {...};`	
	kEncrypt= 1	encryption
	kDecrypt= 2	decryption
	kMacVerify= 3	MAC verification.
	kMacGenerate= 4	MAC generation.
	kWrap= 5	key wrapping

▽

△

	kUnwrap= 6	key unwrapping
	kSigVerify= 7	signature verification
	kSigGenerate= 8	signature generation
Header file:	#include "ara/crypto/common/base_id_types.h"	
Description:	Enumeration of cryptographic transformations.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_10852]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::VolatileTrustedContainer
Derived from:	std::unique_ptr<VolatileTrustedContainer>
Syntax:	`using Uptr = std::unique_ptr<VolatileTrustedContainer>;`
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_10400]{DRAFT} ⌈

Kind:	struct
Symbol:	Uuid
Scope:	namespace ara::crypto
Syntax:	`struct Uuid {...};`
Header file:	#include "ara/crypto/common/uuid.h"
Description:	Definition of Universally Unique Identifier (UUID) type. Independently from internal definition details of this structure, it's size must be 16 bytes and entropy of this ID should be close to 128 bit!

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10801]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptr
Scope:	class ara::crypto::IOInterface
Derived from:	std::unique_ptr<IOInterface>
Syntax:	`using Uptr = std::unique_ptr<IOInterface>;`
Header file:	#include "ara/crypto/common/io_interface.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02109)*

[SWS_CRYPT_10802]{DRAFT} ⌈

Kind:	type alias
Symbol:	Uptrc
Scope:	class ara::crypto::IOInterface
Derived from:	std::unique_ptr<const IOInterface>
Syntax:	`using Uptrc = std::unique_ptr<const IOInterface>;`
Header file:	#include "ara/crypto/common/io_interface.h"
Description:	Unique smart pointer of the constant interface.

⌋(*RS_CRYPTO_02109*)

[SWS_CRYPT_19903]{DRAFT} ⌈

Kind:	type alias
Symbol:	Errc
Scope:	class ara::crypto::SecurityErrorDomain
Derived from:	SecurityErrc
Syntax:	`using Errc = SecurityErrc;`
Header file:	#include "ara/crypto/common/security_error_domain.h"
Description:	security error

⌋(*RS_CRYPTO_02310*)

[SWS_CRYPT_19904]{DRAFT} ⌈

Kind:	type alias
Symbol:	Exception
Scope:	class ara::crypto::SecurityErrorDomain
Derived from:	SecurityException
Syntax:	`using Exception = SecurityException;`
Header file:	#include "ara/crypto/common/security_error_domain.h"
Description:	Alias for the exception base class.

⌋(*SWS_CORE_10934*)

[SWS_CRYPT_10018]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	KeySlotType	
Scope:	namespace ara::crypto	
Underlying type:	std::uint32_t	
Syntax:	`enum class KeySlotType : std::uint32_t {...};`	
Values:	kMachine= 1	machine type key-slot - can be managed by application

▽

△

	kApplication= 2	application exclusive type key-slot
Header file:	#include "ara/crypto/common/base_id_types.h"	
Description:	Enumeration of key-slot types; currently only machine and applicaiton key-slots are defined.	

⌋(*RS_CRYPTO_02004*)

8.5 API Reference

[SWS_CRYPT_10800]{DRAFT} ⌈

Kind:	class
Symbol:	IOInterface
Scope:	namespace ara::crypto
Syntax:	`class IOInterface {...};`
Header file:	#include "ara/crypto/common/io_interface.h"
Description:	Formal interface of an IOInterface is used for saving and loading of security objects. Actual saving and loading should be implemented by internal methods known to a trusted pair of Crypto Provider and Storage Provider. Each object should be uniquely identified by its type and Crypto Object Unique Identifier (COUID). This interface suppose that objects in the container are compressed i.e. have a minimal size optimized for.

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10700]{DRAFT} ⌈

Kind:	class
Symbol:	Serializable
Scope:	namespace ara::crypto
Syntax:	`class Serializable {...};`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Serializable object interface.

⌋(*RS_CRYPTO_02105*)

[SWS_CRYPT_10850]{DRAFT} ⌈

Kind:	class
Symbol:	VolatileTrustedContainer
Scope:	namespace ara::crypto
Syntax:	`class VolatileTrustedContainer {...};`
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"

▽

△

Description:	This explicit interface of a volatile Trusted Container is used for buffering CryptoAPI objects in RAM. This class represents a "smart buffer" in that it provides access to the IOInterface, which can be used for querying meta-data of the buffer content.

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_19905]{DRAFT} ⌈

Kind:	class
Symbol:	SecurityException
Scope:	namespace ara::crypto
Base class:	ara::core::Exception
Syntax:	`class SecurityException : public Exception {...};`
Header file:	#include "ara/crypto/common/security_error_domain.h"
Description:	Exception type thrown for CRYPTO errors.

⌋*(SWS_CORE_10910)*

[SWS_CRYPT_19900]{DRAFT} ⌈

Kind:	class
Symbol:	SecurityErrorDomain
Scope:	namespace ara::crypto
Base class:	ara::core::ErrorDomain
Syntax:	`class SecurityErrorDomain final : public ErrorDomain {...};`
Unique ID:	0x8000'0000'0000'0801
Header file:	#include "ara/crypto/common/security_error_domain.h"
Description:	Security Error Domain class that provides interfaces as defined by ara::core::ErrorDomain such as a name of the Security Error Domain or messages for each error code. This class represents an error domain responsible for all errors that may be reported by public APIs in ara::crypto namespace. .

⌋*(RS_AP_00130)*

[SWS_CRYPT_19951]{DRAFT} ⌈

Kind:	function	
Symbol:	MakeErrorCode(SecurityErrorDomain::Errc code, ara::core::ErrorDomain::SupportDataType data)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (SecurityErrorDomain::Errc code, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	an error code identifier from the SecurityErrc enumeration
	data	supplementary data for the error description

▽

△

Return value:	ara::core::ErrorCode	an instance of ErrorCode created according the arguments
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/security_error_domain.h"	
Description:	Makes Error Code instances from the Security Error Domain. The returned ErrorCode instance always references to SecurityErrorDomain.	

⌋(*RS_CRYPTO_02310*)

[SWS_CRYPT_20099]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadCryptoProvider(const ara::core::InstanceSpecifier &iSpecify)	
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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/entry_point.h"	
Description:	Factory that creates or return existing single instance of specific Crypto Provider. If (providerUid == nullptr) then platform default provider should be loaded.	

⌋(*RS_CRYPTO_02401*)

[SWS_CRYPT_30099]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadKeyStorageProvider()	
Scope:	namespace ara::crypto	
Syntax:	`keys::KeyStorageProvider::Uptr LoadKeyStorageProvider () noexcept;`	
Return value:	ara::crypto::keys::KeyStorage Provider::Uptr	unique smart pointer to loaded Key Storage Provider
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kRuntimeFault	if the Key Storage Provider instance cannot be created
Header file:	#include "ara/crypto/common/entry_point.h"	
Description:	Factory that creates or return existing single instance of the Key Storage Provider.	

⌋(*RS_CRYPTO_02109, RS_CRYPTO_02401*)

[SWS_CRYPT_40099]{DRAFT} ⌈

Kind:	function	
Symbol:	LoadX509Provider()	
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:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kRuntimeFault	if the X.509 Provider cannot be loaded
Header file:	#include "ara/crypto/common/entry_point.h"	
Description:	Factory that creates or return existing single instance of the X.509 Provider. X.509 Provider should use the default Crypto Provider for hashing and signature verification! Therefore when you load the X.509 Provider, in background it loads the default Crypto Provider too.	

⌋(*RS_CRYPTO_02306*)

[SWS_CRYPT_30098]{DRAFT} ⌈

Kind:	function	
Symbol:	GenerateRandomData(std::uint32_t count)	
Scope:	namespace ara::crypto	
Syntax:	`ara::core::Result<ara::core::Vector<ara::core::Byte> > GenerateRandomData (std::uint32_t count) noexcept;`	
Parameters (in):	count	number of random bytes to generate
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	a buffer filled with the generated random sequence
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kBusyResource	if the used RNG is currently out-of-entropy and therefore cannot provide the requested number of random bytes
Header file:	#include "ara/crypto/common/entry_point.h"	
Description:	Return an allocated buffer with a generated random sequence of the requested size.	

⌋(*RS_CRYPTO_02206*)

[SWS_CRYPT_20098]{DRAFT} ⌈

Kind:	function	
Symbol:	GetSecureCounter()	
Scope:	namespace ara::crypto	
Syntax:	`ara::core::Result<SecureCounter> GetSecureCounter () noexcept;`	
Return value:	ara::core::Result< SecureCounter >	a SecureCounter struct made up of the two unsigned 64 bit values (LSQW and MSQW)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kUnsupported	if the Secure Counter is unsupported by the Crypto Stack implementation on this Platform

▽

△

	SecurityErrorDomain::kAccess Violation	if current Actor has no permission to call this routine
Header file:	#include "ara/crypto/common/entry_point.h"	
Description:	Get current value of 128 bit Secure Counter supported by the Crypto Stack. Secure Counter is a non-rollover monotonic counter that ensures incrementation of its value for each following call. The Secure Counter is presented by two 64 bit components: Most Significant Quadword (MSQW) and Least Significant Quadword (LSQW). During normal operation of the Crypto Stack, the MSQW value is fixed (unchangeable) and only LSQW should be incremented. The LSQW counter can be implemented in the "low-power" (always-powered-up) domain of the main CPU, but the MSQW in the Flash/EEPROM storage. But the MSQW must be incremented if the LSQW reaches the maximum value of all ones. Also the MSQW must be incremented during reinitialisation of the whole Crypto Stack (e.g. if the "low-power" supply was interrupted by some reason). Permission to execute this routine is subject of Identity and Access Management control and may be restricted by application manifest!	

⌋*(RS_CRYPTO_02401)*

[SWS_CRYPT_10112]{DRAFT} ⌈

Kind:	function	
Symbol:	HasEarlierVersionThan(const CryptoObjectUid &anotherId)	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`constexpr bool HasEarlierVersionThan (const CryptoObjectUid &anotherId) const noexcept;`	
Parameters (in):	anotherId	another identifier for the comparison
Return value:	bool	true if this identifier was generated earlier than the anotherId
Exception Safety:	noexcept	
Thread Safety:	Reentrant	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Check whether this identifier was generated earlier than the one provided by the argument.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_10113]{DRAFT} ⌈

Kind:	function	
Symbol:	HasLaterVersionThan(const CryptoObjectUid &anotherId)	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`constexpr bool HasLaterVersionThan (const CryptoObjectUid &anotherId) const noexcept;`	
Parameters (in):	anotherId	another identifier for the comparison
Return value:	bool	true if this identifier was generated later than the anotherId
Exception Safety:	noexcept	
Thread Safety:	Reentrant	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Check whether this identifier was generated later than the one provided by the argument.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_10111]{DRAFT} ⌈

Kind:	function	
Symbol:	HasSameSourceAs(const CryptoObjectUid &anotherId)	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`constexpr bool HasSameSourceAs (const CryptoObjectUid &anotherId) const noexcept;`	
Parameters (in):	anotherId	another identifier for the comparison
Return value:	bool	true if both identifiers has common source (identical value of the mGeneratorUid field)
Exception Safety:	noexcept	
Thread Safety:	Reentrant	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Check whether this identifier has a common source with the one provided by the argument.	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_10114]{DRAFT} ⌈

Kind:	function	
Symbol:	IsNil()	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`bool IsNil () const noexcept;`	
Return value:	bool	true if this identifier is "Nil" and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Reentrant	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Check whether this identifier is "Nil".	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_10115]{DRAFT} ⌈

Kind:	function	
Symbol:	SourceIsNil()	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`bool SourceIsNil () const noexcept;`	
Return value:	bool	true if this identifier is "Nil" and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Reentrant	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Check whether this object's generator identifier is "Nil".	

⌋*(RS_CRYPTO_02006)*

[SWS_CRYPT_10810]{DRAFT} ⌈

Kind:	function
Symbol:	~IOInterface()
Scope:	class ara::crypto::IOInterface
Syntax:	`virtual ~IOInterface () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/io_interface.h"
Description:	Destructor.

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10819]{DRAFT} ⌈

Kind:	function	
Symbol:	GetAllowedUsage()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual AllowedUsageFlags GetAllowedUsage () const noexcept=0;`	
Return value:	AllowedUsageFlags	allowed key/seed usage flags
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return actual allowed key/seed usage flags defined by the key slot prototype for this "Actor" and current content of the container. Volatile containers don't have any prototyped restrictions, but can have restrictions defined at run-time for a current instance of object. A value returned by this method is bitwise AND of the common usage flags defined at run-time and the usage flags defined by the UserPermissions prototype for current "Actor". This method is especially useful for empty permanent prototyped containers.	

⌋(*RS_CRYPTO_02008*)

[SWS_CRYPT_10813]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCapacity()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual std::size_t GetCapacity () const noexcept=0;`	
Return value:	std::size_t	capacity of the underlying buffer of this IOInterface (in bytes)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return capacity of the underlying resource.	

⌋(*RS_CRYPTO_02110*)

[SWS_CRYPT_10812]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCryptoObjectType()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual CryptoObjectType GetCryptoObjectType () const noexcept=0;`	
Return value:	CryptoObjectType	the CryptoObjectType stored inside the referenced resource
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return the CryptoObjectType of the object referenced by this IOInterface.	

⌋(*RS_CRYPTO_02110*)

[SWS_CRYPT_10811]{DRAFT} ⌈

Kind:	function	
Symbol:	GetObjectId()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual CryptoObjectUid GetObjectId () const noexcept=0;`	
Return value:	CryptoObjectUid	type of the content stored in the container
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return COUID of an object stored to this IOInterface. If the container is empty then this method returns CryptoObjectType::KUndefined. Unambiguous identification of a crypto object requires both components: CryptoObjectUid and CryptoObjectType.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10817]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPayloadSize()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual std::size_t GetPayloadSize () const noexcept=0;`	
Return value:	std::size_t	size of an object payload stored in the underlying buffer of this IOInterface (in bytes)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return size of an object payload stored in the underlying buffer of this IOInterface. If the container is empty then this method returns 0. Returned value does not take into account the object's meta-information properties, but their size is fixed and common for all crypto objects independently from their actual type. space for an object's meta-information automatically, according to their implementation details.	

⌋(*RS_CRYPTO_02109*)

[SWS_CRYPT_10822]{DRAFT} ⌈

Kind:	function	
Symbol:	GetPrimitiveId()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual CryptoAlgId GetPrimitiveId () const noexcept=0;`	
Return value:	CryptoAlgId	the binary Crypto Primitive ID
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Get vendor specific ID of the primitive.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_10818]{DRAFT} ⌈

Kind:	function	
Symbol:	GetTypeRestriction()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual CryptoObjectType GetTypeRestriction () const noexcept=0;`	
Return value:	CryptoObjectType	an object type of allowed content (CryptoObjectType::kUndefined means without restriction)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return content type restriction of this IOInterface. If KeySlotPrototypeProps::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.	

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02110)*

[SWS_CRYPT_10816]{DRAFT} ⌈

Kind:	function	
Symbol:	IsObjectExportable()	
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:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return the "exportable" attribute of an object stored to the container. The exportability of an object doesn't depend from the volatility of its container.	

⌋*(RS_CRYPTO_02109)*

[SWS_CRYPT_10815]{DRAFT} ⌈

Kind:	function	
Symbol:	IsObjectSession()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual bool IsObjectSession () const noexcept=0;`	
Return value:	bool	true if the object referenced by this IOInterface has set the "session" attribute
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return the "session" (or "temporary") attribute of an object as set e.g. by KeyDerivationFunctionCtx::DeriveKey(). A "session" object can be stored to a VolatileTrustedContainer only! If this IOInterface is linked to a KeySlot this returns always false.	

⌋*(RS_CRYPTO_02109)*

[SWS_CRYPT_10814]{DRAFT} ⌈

Kind:	function	
Symbol:	IsVolatile()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual bool IsVolatile () const noexcept=0;`	
Return value:	bool	true if the container has a volatile nature (i.e. "temporary" or "in RAM") or false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Return volatility of the the underlying buffer of this IOInterface. A "session" object can be stored to a "volatile" container only. A content of a "volatile" container will be destroyed together with the interface instance.	

⌋*(RS_CRYPTO_02109)*

[SWS_CRYPT_10823]{DRAFT} ⌈

Kind:	function	
Symbol:	IsValid()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual bool IsValid () const noexcept=0;`	
Return value:	bool	true if the underlying resource can be valid, false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Get whether the underlying KeySlot is valid. An IOInterface is invalidated if the underlying resource has been modified after the IOInterface has been opened.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_10821]{DRAFT} ⌈

Kind:	function	
Symbol:	IsWritable()	
Scope:	class ara::crypto::IOInterface	
Syntax:	`virtual bool IsWritable () const noexcept=0;`	
Return value:	bool	true if the underlying resource can be written
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Get whether the underlying KeySlot is writable - if this IOInterface is linked to a VolatileTrusted Container always return true.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30202]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const IOInterface &other)	
Scope:	class ara::crypto::IOInterface	
Syntax:	`IOInterface& operator= (const IOInterface &other)=default;`	
Parameters (in):	other	the other instance
Return value:	IOInterface &	*this, containing the contents of other
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Copy-assign another IOInterface to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_30203]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(IOInterface &&other)	
Scope:	class ara::crypto::IOInterface	
Syntax:	`IOInterface& operator= (IOInterface &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	IOInterface &	*this, containing the contents of other
Header file:	#include "ara/crypto/common/io_interface.h"	
Description:	Move-assign another IOInterface to this instance.	

⌋*(RS_CRYPTO_02004)*

[SWS_CRYPT_10150]{DRAFT} ⌈

Kind:	function	
Symbol:	operator==(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator== (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if all members' values of lhs is equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Comparison operator "equal" for CryptoObjectUid operands.	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10151]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator< (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is less than rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Comparison operator "less than" for CryptoObjectUid operands.	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10152]{DRAFT} ⌈

Kind:	function	
Symbol:	operator>(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator> (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is greater than rhs, and false otherwise
Exception Safety:	noexcept	

▽

△

Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Description:	Comparison operator "greater than" for CryptoObjectUid operands.

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10153]{DRAFT} ⌈

Kind:	function	
Symbol:	operator!=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator!= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if at least one member of lhs has a value not equal to correspondent member of rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Comparison operator "not equal" for CryptoObjectUid operands.	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10154]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator<= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is less than or equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Comparison operator "less than or equal" for CryptoObjectUid operands.	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10155]{DRAFT} ⌈

Kind:	function	
Symbol:	operator>=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator>= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is greater than or equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/crypto_object_uid.h"	
Description:	Comparison operator "greater than or equal" for CryptoObjectUid operands.	

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10451]{DRAFT} ⌈

Kind:	function	
Symbol:	operator==(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator== (const Uuid &lhs, const Uuid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/uuid.h"	
Description:	Comparison operator "equal" for Uuid operands.	

⌋*(RS_CRYPTO_02112)*

[SWS_CRYPT_10452]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator< (const Uuid &lhs, const Uuid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is less than rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

Header file:	#include "ara/crypto/common/uuid.h"
Description:	Comparison operator "less than" for Uuid operands.

⌋(*RS_CRYPTO_02112*)

[SWS_CRYPT_10453]{DRAFT} ⌈

Kind:	function	
Symbol:	operator>(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator> (const Uuid &lhs, const Uuid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is greater than rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/uuid.h"	
Description:	Comparison operator "greater than" for Uuid operands.	

⌋(*RS_CRYPTO_02112*)

[SWS_CRYPT_10454]{DRAFT} ⌈

Kind:	function	
Symbol:	operator!=(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator!= (const Uuid &lhs, const Uuid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is not equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/uuid.h"	
Description:	Comparison operator "not equal" for Uuid operands.	

⌋(*RS_CRYPTO_02112*)

[SWS_CRYPT_10455]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<=(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator<= (const Uuid &lhs, const Uuid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is less than or equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/uuid.h"	
Description:	Comparison operator "less than or equal" for Uuid operands.	

⌋*(RS_CRYPTO_02112)*

[SWS_CRYPT_10456]{DRAFT} ⌈

Kind:	function	
Symbol:	operator>=(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`constexpr bool operator>= (const Uuid &lhs, const Uuid &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if a binary representation of lhs is greater than or equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/uuid.h"	
Description:	Comparison operator "greater than or equal" for Uuid operands.	

⌋*(RS_CRYPTO_02112)*

[SWS_CRYPT_19954]{DRAFT} ⌈

Kind:	function	
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)	
Scope:	class ara::crypto::SecurityErrorDomain	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const override;`	
Parameters (in):	errorCode	an error code identifier from the SecurityErrc enumeration
Return value:	None	
Header file:	#include "ara/crypto/common/security_error_domain.h"	
Description:	throws exception of error code	

⌋*(RS_CRYPTO_02310)*

[SWS_CRYPT_19902]{DRAFT} ⌈

Kind:	function
Symbol:	SecurityErrorDomain()
Scope:	class ara::crypto::SecurityErrorDomain
Syntax:	`constexpr SecurityErrorDomain () noexcept;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/security_error_domain.h"
Description:	Ctor of the SecurityErrorDomain.

⌋*(RS_CRYPTO_02310)*

[SWS_CRYPT_19950]{DRAFT} ⌈

Kind:	function	
Symbol:	Name()	
Scope:	class ara::crypto::SecurityErrorDomain	
Syntax:	`const char* Name () const noexcept override;`	
Return value:	const char *	"Security" text
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/security_error_domain.h"	
Description:	returns Text "Security"	

⌋*(RS_CRYPTO_02310)*

[SWS_CRYPT_19953]{DRAFT} ⌈

Kind:	function	
Symbol:	Message(ara::core::ErrorDomain::CodeType errorCode)	
Scope:	class ara::crypto::SecurityErrorDomain	
Syntax:	`const char* Message (ara::core::ErrorDomain::CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	an error code identifier from the SecurityErrc enumeration
Return value:	const char *	message text of error code
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/security_error_domain.h"	
Description:	Translate an error code value into a text message.	

⌋*(RS_CRYPTO_02310)*

[SWS_CRYPT_10710]{DRAFT} ⌈

Kind:	function
Symbol:	~Serializable()
Scope:	class ara::crypto::Serializable

▽

△

Syntax:	`virtual ~Serializable () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Destructor.

⌋(*RS_CRYPTO_02004, RS_CRYPTO_02302*)

[SWS_CRYPT_10711]{DRAFT} ⌈

Kind:	function	
Symbol:	ExportPublicly(FormatId formatId=kFormatDefault)	
Scope:	class ara::crypto::Serializable	
Syntax:	`virtual ara::core::Result<ara::core::Vector<ara::core::Byte> > ExportPublicly (FormatId formatId=kFormatDefault) const noexcept=0;`	
Parameters (in):	formatId	the Crypto Provider specific identifier of the output format
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	a buffer with the serialized object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Errors:	SecurityErrorDomain::kInsufficientCapacity	if (output.empty() == false), but it's capacity is less than required
	SecurityErrorDomain::kUnknownIdentifier	if an unknown format ID was specified
	SecurityErrorDomain::kUnsupportedFormat	if the specified format ID is not supported for this object type
Header file:	#include "ara/crypto/common/serializable.h"	
Description:	Serialize itself publicly.	

⌋(*RS_CRYPTO_02112*)

[SWS_CRYPT_10712]{DRAFT} ⌈

Kind:	function	
Symbol:	ExportPublicly(FormatId formatId=kFormatDefault)	
Scope:	class ara::crypto::Serializable	
Syntax:	`template <typename Alloc = <implementation-defined>> ara::core::Result<ByteVector<Alloc> > ExportPublicly (FormatId formatId=kFormatDefault) const noexcept;`	
Template param:	Alloc	custom allocator type of the output container
Parameters (in):	formatId	the Crypto Provider specific identifier of the output format
Return value:	ara::core::Result< ByteVector< Alloc > >	pre-reserved managed container for the serialization output
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

Errors:	SecurityErrorDomain::kInsufficient Capacity	if capacity of the output buffer is less than required
	SecurityErrorDomain::kUnknown Identifier	if an unknown format ID was specified
	SecurityErrorDomain::kUnsupported Format	if the specified format ID is not supported for this object type
Header file:	#include "ara/crypto/common/serializable.h"	
Description:	Serialize itself publicly. This method sets the size of the output container according to actually saved value!	

⌋(*RS_CRYPTO_02112*)

[SWS_CRYPT_30204]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const Serializable &other)	
Scope:	class ara::crypto::Serializable	
Syntax:	`Serializable& operator= (const Serializable &other)=default;`	
Parameters (in):	other	the other instance
Return value:	Serializable &	*this, containing the contents of other
Header file:	#include "ara/crypto/common/serializable.h"	
Description:	Copy-assign another Serializable to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30205]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(Serializable &&other)	
Scope:	class ara::crypto::Serializable	
Syntax:	`Serializable& operator= (Serializable &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	Serializable &	*this, containing the contents of other
Header file:	#include "ara/crypto/common/serializable.h"	
Description:	Move-assign another Serializable to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10851]{DRAFT} ⌈

Kind:	function
Symbol:	~VolatileTrustedContainer()
Scope:	class ara::crypto::VolatileTrustedContainer
Syntax:	`virtual ~VolatileTrustedContainer () noexcept=default;`

▽

△

Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"
Description:	Destructor.

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10853]{DRAFT} ⌈

Kind:	function	
Symbol:	GetIOInterface()	
Scope:	class ara::crypto::VolatileTrustedContainer	
Syntax:	`virtual IOInterface& GetIOInterface () const noexcept=0;`	
Return value:	IOInterface &	a reference to the IOInterface of this container
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"	
Description:	Retrieve the IOInterface used for importing/exporting objects into this container.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30206]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const VolatileTrustedContainer &other)	
Scope:	class ara::crypto::VolatileTrustedContainer	
Syntax:	`VolatileTrustedContainer& operator= (const VolatileTrustedContainer &other)=default;`	
Parameters (in):	other	the other instance
Return value:	VolatileTrustedContainer &	*this, containing the contents of other
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"	
Description:	Copy-assign another VolatileTrustedContainer to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_30207]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(VolatileTrustedContainer &&other)	
Scope:	class ara::crypto::VolatileTrustedContainer	
Syntax:	`VolatileTrustedContainer& operator= (VolatileTrustedContainer &&other)=default;`	
Parameters (in):	other	the other instance
Return value:	VolatileTrustedContainer &	*this, containing the contents of other
Header file:	#include "ara/crypto/common/volatile_trusted_container.h"	
Description:	Move-assign another VolatileTrustedContainer to this instance.	

⌋(*RS_CRYPTO_02004*)

[SWS_CRYPT_10411]{DRAFT} ⌈

Kind:	function	
Symbol:	IsNil()	
Scope:	struct ara::crypto::Uuid	
Syntax:	`bool IsNil () const noexcept;`	
Return value:	bool	true if this identifier is "Nil" and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/uuid.h"	
Description:	Check whether this identifier is the "Nil UUID" (according to RFC4122).	

⌋(*RS_CRYPTO_02005*)

[SWS_CRYPT_13000]{DRAFT} ⌈

Kind:	variable
Symbol:	kAlgIdUndefined
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId kAlgIdUndefined = 0u;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Algorithm ID is undefined. Also this value may be used in meanings: Any or Default algorithm, None of algorithms. Effective values of Crypto Algorithm IDs are specific for concrete Crypto Stack implementation. But the zero value is reserved for especial purposes, that can differ depending from a usage context. This group defines a few constant names of the single zero value, but semantically they have different meaning specific for concrete application of the constant.

⌋(*RS_CRYPTO_02107*)

[SWS_CRYPT_13001]{DRAFT} ⌈

Kind:	variable
Symbol:	kAlgIdAny
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId kAlgIdAny = kAlgIdUndefined;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Any Algorithm ID is allowed.

⌋(*RS_CRYPTO_02107*)

[SWS_CRYPT_13002]{DRAFT} ⌈

Kind:	variable
Symbol:	kAlgIdDefault
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId kAlgIdDefault = kAlgIdUndefined;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Default Algorithm ID (in current context/primitive).

⌋(*RS_CRYPTO_02107*)

[SWS_CRYPT_13003]{DRAFT} ⌈

Kind:	variable
Symbol:	kAlgIdNone
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId kAlgIdNone = kAlgIdUndefined;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	None of Algorithm ID (i.e. an algorithm definition is not applicable).

⌋(*RS_CRYPTO_02107*)

[SWS_CRYPT_13102]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDataDecryption
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDataDecryption = 0x0002;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The key/seed can be used for data decryption initialization (applicable to symmetric and asymmetric algorithms).

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13101]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDataEncryption
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDataEncryption = 0x0001;`
Header file:	#include "ara/crypto/common/base_id_types.h"

▽

△

Description:	The key/seed can be used for data encryption initialization (applicable to symmetric and asymmetric algorithms).

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13113]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedDataDecryption
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedDataDecryption = kAllowDataDecryption << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for data decryption.

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13112]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedDataEncryption
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedDataEncryption = kAllowDataEncryption << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for data encryption.

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13117]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedRngInit
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedRngInit = kAllowRngInit << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for seeding of a RandomGeneratorContext.

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13121]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedExactModeOnly
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedExactModeOnly = kAllowExactModeOnly << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Restrict usage of derived objects to specified operation mode only. A derived seed or symmetric key can be used only for the mode directly specified by Key::AlgId.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13118]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedKdfMaterial
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedKdfMaterial = kAllowKdfMaterial << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used as a RestrictedUseObject for slave-keys derivation via a Key Derivation Function (KDF).

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13122]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowKdfMaterialAnyUsage
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowKdfMaterialAnyUsage = kAllowKdfMaterial | kAllowDerivedDataEncryption | kAllowDerivedDataDecryption | kAllowDerivedSignature | kAllowDerivedVerification | kAllowDerivedKeyDiversify | kAllowDerivedRngInit | kAllowDerivedKdfMaterial | kAllowDerivedKeyExporting | kAllowDerivedKeyImporting;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Allow usage of the object as a key material for KDF and any usage of derived objects. The seed or symmetric key can be used as a RestrictedUseObject for a Key Derivation Function (KDF) and the derived "slave" keys can be used without limitations.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13116]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedKeyDiversify
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedKeyDiversify = kAllowKeyDiversify << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for slave-keys diversification.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13119]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedKeyExporting
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedKeyExporting = kAllowKeyExporting << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used as a "transport" one for Key-Wrap transformation.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13120]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedKeyImporting
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedKeyImporting = kAllowKeyImporting << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used as a "transport" one for Key-Unwrap transformation.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13114]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedSignature
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedSignature = kAllowSignature << 16;`

▽

△

Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for MAC/HMAC production.

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13115]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowDerivedVerification
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowDerivedVerification = kAllowVerification << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for MAC/HMAC verification.

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13111]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowExactModeOnly
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowExactModeOnly = 0x8000;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The key can be used only for the mode directly specified by Key::AlgId.

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13108]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowKdfMaterial
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowKdfMaterial = 0x0080;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The object can be used as an input key material to KDF. The seed or symmetric key can be used as a RestrictedUseObject for slave-keys derivation via a Key Derivation Function (KDF).

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_13105]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowKeyAgreement
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowKeyAgreement = 0x0010;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The seed or asymmetric key can be used for key-agreement protocol execution.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13106]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowKeyDiversify
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowKeyDiversify = 0x0020;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The seed or symmetric key can be used for slave-keys diversification.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13109]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowKeyExporting
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowKeyExporting = 0x0100;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The key can be used as "transport" one for Key-Wrap or Encapsulate transformations (applicable to symmetric and asymmetric keys).

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13110]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowKeyImporting
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowKeyImporting = 0x0200;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The key can be used as "transport" one for Key-Unwrap or Decapsulate transformations (applicable to symmetric and asymmetric keys).

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13100]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowPrototypedOnly
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowPrototypedOnly = 0;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	This group contains list of constant 1-bit values predefined for Allowed Usage flags. The key/seed usage will be fully specified by a key slot prototype (the object can be used only after reloading from the slot).

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13107]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowRngInit
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowRngInit = 0x0040;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The seed or symmetric key can be used for seeding of a RandomGeneratorCtx.

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13103]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowSignature
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags kAllowSignature = 0x0004;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The key/seed can be used for digital signature or MAC/HMAC production (applicable to symmetric and asymmetric algorithms).

⌋*(RS_CRYPTO_02111)*

[SWS_CRYPT_13104]{DRAFT} ⌈

Kind:	variable
Symbol:	kAllowVerification
Scope:	namespace ara::crypto
Type:	const AllowedUsageFlags

▽

△

Syntax:	`const AllowedUsageFlags kAllowVerification = 0x0008;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The key/seed can be used for digital signature or MAC/HMAC verification (applicable to symmetric and asymmetric algorithms).

⌋(*RS_CRYPTO_02111*)

[SWS_CRYPT_10102]{DRAFT} ⌈

Kind:	variable
Symbol:	mVersionStamp
Scope:	struct ara::crypto::CryptoObjectUid
Type:	std::uint64_t
Syntax:	`std::uint64_t mVersionStamp = 0u;`
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Description:	Sequential value of a steady timer or simple counter, representing version of correspondent Crypto Object.

⌋(*RS_CRYPTO_02006*)

[SWS_CRYPT_30002]{DRAFT} ⌈

Kind:	variable
Symbol:	mLSQW
Scope:	struct ara::crypto::SecureCounter
Type:	std::uint64_t
Syntax:	`std::uint64_t mLSQW;`
Header file:	#include "ara/crypto/common/entry_point.h"
Description:	least significant 64 bits

⌋(*RS_CRYPTO_02401*)

[SWS_CRYPT_30003]{DRAFT} ⌈

Kind:	variable
Symbol:	mMSQW
Scope:	struct ara::crypto::SecureCounter
Type:	std::uint64_t
Syntax:	`std::uint64_t mMSQW;`
Header file:	#include "ara/crypto/common/entry_point.h"
Description:	most significant 64 bits

⌋(*RS_CRYPTO_02401*)

[SWS_CRYPT_10750]{DRAFT} ⌈

Kind:	variable
Symbol:	kFormatDefault
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId kFormatDefault = 0;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Default serialization format.

⌋(*RS_CRYPTO_02004, RS_CRYPTO_02302*)

[SWS_CRYPT_10752]{DRAFT} ⌈

Kind:	variable
Symbol:	kFormatDerEncoded
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId kFormatDerEncoded = 2;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Export DER-encoded value of an object.

⌋(*RS_CRYPTO_02004, RS_CRYPTO_02302*)

[SWS_CRYPT_10753]{DRAFT} ⌈

Kind:	variable
Symbol:	kFormatPemEncoded
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId kFormatPemEncoded = 3;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Export PEM-encoded value of an object.

⌋(*RS_CRYPTO_02004, RS_CRYPTO_02302*)

[SWS_CRYPT_10751]{DRAFT} ⌈

Kind:	variable
Symbol:	kFormatRawValueOnly
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId kFormatRawValueOnly = 1;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Export only raw value of an object.

⌋(*RS_CRYPTO_02004, RS_CRYPTO_02302*)

[SWS_CRYPT_10412]{DRAFT} ⌈

Kind:	variable
Symbol:	mQwordLs
Scope:	struct ara::crypto::Uuid
Type:	std::uint64_t
Syntax:	`std::uint64_t mQwordLs = 0u;`
Header file:	#include "ara/crypto/common/uuid.h"
Description:	Less significant QWORD.

⌋*(RS_CRYPTO_02005)*

[SWS_CRYPT_10413]{DRAFT} ⌈

Kind:	variable
Symbol:	mQwordMs
Scope:	struct ara::crypto::Uuid
Type:	std::uint64_t
Syntax:	`std::uint64_t mQwordMs = 0u;`
Header file:	#include "ara/crypto/common/uuid.h"
Description:	Most significant QWORD.

⌋*(RS_CRYPTO_02005)*

9 Service Interfaces

This chapter lists all provided and required service interfaces of the `For instance Diagnostic Management`.

Tables are generated out of 'arxml' folder content.

9.1 Type definitions

No types are defined for service interfaces.

9.2 Provided Service Interfaces

No service interfaces are provided.

9.3 Required Service Interfaces

No service interfaces are required.

9.4 Application Errors

No application errors are defined.

A Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Chapter is generated.

Class	**AdaptiveApplicationSwComponentType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			
Note	This meta-class represents the ability to support the formal modeling of application software on the AUTOSAR adaptive platform. Consequently, it shall only be used on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=AdaptiveApplicationSwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
internalBehavior	AdaptiveSwcInternal Behavior	0..1	aggr	This aggregation represents the internal behavior of the AdaptiveApplicationSwComponentType for the AUTOSAR adaptive platform. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior.shortName, internal Behavior.variationPoint.shortLabel atp.Status=draft vh.latestBindingTime=preCompileTime

Table A.1: AdaptiveApplicationSwComponentType

Class	**CryptoCertificate**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class represents the ability to model a cryptographic certificate. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
isPrivate	Boolean	0..1	attr	This attribute controls the possibility to access the content of the CryptoCertificateSlot by Find() interfaces of the X509 Provider.

Table A.2: CryptoCertificate

Class	**CryptoCertificateInterface**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign
Note	This meta-class provides the ability to define a PortInterface for a CryptoCertificate. **Tags:** atp.Status=draft atp.recommendedPackage=CryptoInterfaces
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *CryptoInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*

▽

△

Class	**CryptoCertificateInterface**			
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.
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.3: CryptoCertificateInterface

Class	**CryptoCertificateToCryptoKeySlotMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class represents the ability to define a mapping between a CryptoKeySlot and a Crypto Certificate. **Tags:**atp.Status=draft			
Base	*ARObject*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
crypto Certificate	CryptoCertificate	1	ref	This reference represents the mapped cryptoCertificate. **Tags:**atp.Status=draft
cryptoKeySlot	CryptoKeySlot	0..2	ref	This reference represents the mapped cryptoKeySlot. **Tags:**atp.Status=draft

Table A.4: CryptoCertificateToCryptoKeySlotMapping

Class	**CryptoKeySlot**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class represents the ability to define a concrete key to be used for a crypto operation. **Tags:** atp.ManifestKind=MachineManifest atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
allocateShadow Copy	Boolean	0..1	attr	This attribute defines whether a shadow copy of this Key Slot shall be allocated to enable rollback of a failed Key Slot update campaign (see interface BeginTransaction).
cryptoAlgId	String	0..1	attr	This attribute defines a crypto algorithm restriction (kAlgId Any means without restriction). The algorithm can be specified partially: family & length, mode, padding. 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.

▽

△

Class	**CryptoKeySlot**			
cryptoObject Type	CryptoObjectTypeEnum	0..1	attr	Object type that can be stored in the slot. If this field contains "Undefined" then mSlotCapacity must be provided and larger then 0.
keySlotAllowed Modification	CryptoKeySlotAllowed Modification	0..1	aggr	Restricts how this keySlot may be used **Tags:**atp.Status=draft
keySlotContent AllowedUsage	CryptoKeySlotContent AllowedUsage	*	aggr	Restriction of allowed usage of a key stored to the slot. **Tags:**atp.Status=draft
slotCapacity	PositiveInteger	0..1	attr	Capacity of the slot in bytes to be reserved by the stack vendor. One use case is to define this value in case that the cryptoObjectType is undefined and the slot size can not be deduced from cryptoObjectType and cryptoAlgId. "0" means slot size can be deduced from cryptoObject Type and cryptoAlgId.
slotType	CryptoKeySlotType Enum	0..1	attr	This attribute defines whether the keySlot is exclusively used by the Application; or whether it is used by Stack Services and managed by a Key Manager Application.

Table A.5: CryptoKeySlot

Class	**CryptoKeySlotInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign			
Note	This meta-class provides the ability to define a PortInterface for Crypto Key Slots. **Tags:** atp.Status=draft atp.recommendedPackage=CryptoInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *CryptoInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allocateShadow Copy	Boolean	0..1	attr	This attribute defines whether a shadow copy of this Key Slot shall be allocated to enable rollback of a failed Key Slot update campaign (see interface BeginTransaction).
cryptoAlgId	String	0..1	attr	This attribute defines a crypto algorithm restriction (kAlgId Any means without restriction). The algorithm can be specified partially: family & length, mode, padding. Future Crypto Providers can support some crypto algorithms that are not well known/ standardized today, therefore AUTOSAR doesn't provide a concrete list of crypto algorithms' identifiers and doesn't suppose usage of numerical identifiers. Instead of this a provider supplier should provide string names of supported algorithms in accompanying documentation. The name of a crypto algorithm shall follow the rules defined in the specification of cryptography for Adaptive Platform.
cryptoObject Type	CryptoObjectTypeEnum	0..1	attr	Object type that can be stored in the slot. If this field contains "Undefined" then mSlotCapacity must be provided and larger then 0
keySlotAllowed Modification	CryptoKeySlotAllowed Modification	0..1	aggr	Restricts how this keySlot may be used **Tags:**atp.Status=draft
keySlotContent AllowedUsage	CryptoKeySlotContent AllowedUsage	*	aggr	Restriction of allowed usage of a key stored to the slot. **Tags:**atp.Status=draft

▽

△

Class	CryptoKeySlotInterface			
slotCapacity	PositiveInteger	0..1	attr	Capacity of the slot in bytes to be reserved by the stack vendor. One use case is to define this value in case that the cryptoObjectType is undefined and the slot size can not be deduced from cryptoObjectType and cryptoAlgId. "0" means slot size can be deduced from cryptoObject Type and cryptoAlgId.
slotType	CryptoKeySlotType Enum	0..1	attr	This attribute defines whether the keySlot is exclusively used by the Application; or whether it is used by Stack Services and managed by a Key Manager Application.

Table A.6: CryptoKeySlotInterface

Class	**CryptoKeySlotToPortPrototypeMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class represents the ability to define a mapping between a CryptoKeySlot on deployment level to a given PortPrototype that is typed by a CryptoKeySlotInterface. **Tags:** atp.Status=draft atp.recommendedPackage=CryptoKeySlotToPortPrototypeMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
keySlot	CryptoKeySlot	1	ref	This reference represents the mapped CryptoKeySlot. **Tags:**atp.Status=draft
portPrototype	PortPrototype	0..1	iref	This reference represents the mapped PortPrototype. **Tags:**atp.Status=draft **InstanceRef implemented by:**PortPrototypeIn ExecutableInstanceRef
process	Process	1	ref	This reference represents the process required as context for the mapping. **Tags:**atp.Status=draft

Table A.7: CryptoKeySlotToPortPrototypeMapping

Class	**CryptoProvider**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	CryptoProvider implements cryptographic primitives (algorithms) supported by the stack. Implementation of this component may be software or hardware based (HSM/TPM). **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
cryptoProvider Documentation	Documentation	0..1	ref	Documentation of the CryptoProvider that describes the implemented cryptographic primitives. **Tags:**atp.Status=draft

▽

△

Class	CryptoProvider			
keySlot	CryptoKeySlot	*	aggr	This aggregation represents the key slots that are allocated by the CryptoProvider. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=keySlot.shortName atp.Status=draft

Table A.8: CryptoProvider

Class	**CryptoProviderInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CryptoDesign			
Note	This meta-class provides the ability to define a PortInterface for a CryptoProvider. **Tags:** atp.Status=draft atp.recommendedPackage=CryptoInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *CryptoInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.9: CryptoProviderInterface

Class	**CryptoProviderToPortPrototypeMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class represents the ability to define a mapping between a CryptoProvider on deployment level to a given PortPrototype that is typed by a CryptoProviderInterface. **Tags:** atp.Status=draft atp.recommendedPackage=CryptoProviderToPortPrototypeMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
cryptoProvider	CryptoProvider	1	ref	This reference represents the mapped cryptoProvider. **Tags:**atp.Status=draft
portPrototype	PortPrototype	0..1	iref	This reference represents the mapped PortPrototype. **Tags:**atp.Status=draft **InstanceRef implemented by:**PortPrototypeIn ExecutableInstanceRef
process	Process	1	ref	This reference represents the process required as context for the mapping. **Tags:**atp.Status=draft

Table A.10: 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*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
algorithmFamily	CryptoCertificate AlgorithmFamilyEnum	0..1	attr	This attribute represents a description of the family of crypto algorithm used to generate public key and signature of the cryptographic certificate.
format	CryptoCertificateFormat Enum	0..1	attr	This attribute can be used to provide information about the format used to create the certificate
maximum Length	PositiveInteger	0..1	attr	This attribute represents the ability to define the maximum length of the certificate.
nextHigher Certificate	CryptoService Certificate	0..1	ref	The reference identifies the next higher certificate in the certificate chain.

Table A.11: CryptoServiceCertificate

Class	Process			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class provides information required to execute the referenced executable. **Tags:** atp.Status=draft atp.recommendedPackage=Processes			
Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
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. **Tags:**atp.Status=draft
deterministic Client	DeterministicClient	0..1	ref	This reference adds further execution characteristics for deterministic clients. **Tags:**atp.Status=draft
executable	Executable	0..1	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft

▽

△

Class	**Process**			
securityEvent	SecurityEventDefinition	*	ref	The reference identifies the collection of SecurityEvents that can be reported by the enclosing SoftwareCluster. **Stereotypes:** atpSplitable; atpUriDef **Tags:** atp.Splitkey=securityEvent atp.Status=draft
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations. **Tags:**atp.Status=draft

Table A.12: 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*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
required Interface	PortInterface	0..1	tref	The interface that this port requires. **Stereotypes:** isOfType

Table A.13: 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

C History of Constraints and Specification Items

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

C.1 Constraint and Specification Item History of this document according to AUTOSAR Release yy-mm

...