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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	19-03

Document Change History

Date	Release	Changed by	Description
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

Table of Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Adaptive Crypto Stack as part of the functional cluster Security Management of the AUTOSAR Adaptive Platform.
The Crypto Stack offers applications a standardized interface to cryptographic operations. The Crypto Stack realizes the APIs and manages actual implementations of operations, as well as management functionality handling configuration and brokering.

2 Acronyms and Abbreviations

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

Abbreviation / Acronym:	**Description:**
COUID	Cryptographic Object Unique Identifier
CRL	Certificate Revocation Lists
CSR	Certificate Signing Request
DER	Distinguished Encoding Rules
DH	Diffie-Hellman (key exchange method)
ECC	Elliptic Curve Cryptography
HSM	Hardware Security Module
IPC	Inter-Process Communication
IV	Initialization Vector
MAC	Message Authentication Code
OCSP	On-line Certificate Status Protocol
PEM	Privacy-Enhanced Mail
PKI	Public Key Infrastructure
TPM	Trusted Platform Module
UID	Unique Identifier

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

[2] Requirements on Security Management for Adaptive Platform
AUTOSAR_RS_SecurityManagement

4 Constraints and assumptions

4.1 Limitations

The current version of this document is missing some functionality that was available in the AUTOSAR Classic Platform:

- **Secure Counter**
  There is currently no API available to access secure counter primitives that an implementation may provide.

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

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

- **X.509 certificates support**
  Crypto API doesn't provide complete specification of the X.509 certificates management 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.

- **Memory management**
  Now only Crypto Provider supports the safety-aligned memory management concept suitable for real-time applications. Up to the next release this concept will be extended for X.509 Provider too.
  Application of any memory management mechanisms specific for support of asynchronous calls (like `std::future`) is in scope a developer responsibility.

- **Formats of cryptographic objects**
  Current version of Crypto API 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 Crypto API. For other cryptographic objects an implementation can support only "raw" formats. Following extension of the Crypto API by unified interfaces for encoding/decoding of complex objects to standard formats is planned for the next release of this specification.

- **Key slots modeling**
  Now Crypto API defines some structures that should be produced as a result of the key slots modeling process. But the whole concept of the key slots modeling is not finished yet. Therefore Key Storage API can be updated slightly for next release in order to extend support of the `ara::core::InstanceSpecifier` type as one of mechanisms for the Logical Key Slot identification.
- **Functional specification**
  Detailed functional specification (chapter 7) is not prepared yet and it will be elaborated for next Autosar AP release.
- **Depth of inheritance**
  The performance of the inheritance tree design applied for the Crypto Provider interfaces is still subject to further investigation. Therefore a redesign of APIs defined in namespace `ara::crypto::cryp` may be executed for next Autosar release, in order to achieve a very limited inheritance depth (or completely "flattened" API design).

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other functional clusters

There are currently no dependencies to other functional clusters.

6 Requirements Tracing

The following tables reference the requirements specified in [2] 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_CRYPTO_02001]	The Crypto Stack shall conceal symmetric keys from the users	[SWS_CRYPT_20733] [SWS_CRYPT_20760] [SWS_CRYPT_23800] [SWS_CRYPT_23911]
[RS_CRYPTO_02002]	The Crypto Stack shall conceal asymmetric private keys from the users	[SWS_CRYPT_20733] [SWS_CRYPT_20760] [SWS_CRYPT_22500] [SWS_CRYPT_22611]
[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_22611] [SWS_CRYPT_23911]
[RS_CRYPTO_02004]	The Crypto Stack shall support secure storage of cryptographic artifacts	[SWS_CRYPT_10016] [SWS_CRYPT_10800] [SWS_CRYPT_10803] [SWS_CRYPT_10811] [SWS_CRYPT_10818] [SWS_CRYPT_10820] [SWS_CRYPT_10821] [SWS_CRYPT_20517] [SWS_CRYPT_30003] [SWS_CRYPT_30004] [SWS_CRYPT_30112] [SWS_CRYPT_30123] [SWS_CRYPT_30124] [SWS_CRYPT_30125] [SWS_CRYPT_30126] [SWS_CRYPT_30200] [SWS_CRYPT_30211]
[RS_CRYPTO_02005]	The Crypto Stack shall support unique identification of cryptographic objects	[SWS_CRYPT_10100] [SWS_CRYPT_20413] [SWS_CRYPT_20514] [SWS_CRYPT_20515] [SWS_CRYPT_30102] [SWS_CRYPT_30112] [SWS_CRYPT_30500]
[RS_CRYPTO_02006]	The Crypto Stack shall support a version control mechanism and distinguish “versions” and “origin sources” of cryptographic objects	[SWS_CRYPT_10100] [SWS_CRYPT_10101] [SWS_CRYPT_10102] [SWS_CRYPT_10111] [SWS_CRYPT_10112] [SWS_CRYPT_10113] [SWS_CRYPT_10114]

Requirement	Description	Satisfied by
[RS_CRYPTO_02007]	The Crypto Stack shall provide means for secure handling of “secret seeds”	[SWS_CRYPT_20723] [SWS_CRYPT_21311] [SWS_CRYPT_21411] [SWS_CRYPT_21514] [SWS_CRYPT_21515] [SWS_CRYPT_21811] [SWS_CRYPT_21812] [SWS_CRYPT_23000] [SWS_CRYPT_24015]
[RS_CRYPTO_02008]	The Crypto Stack shall support restrictions of the allowed usage scope for keys and “secret seeds”	[SWS_CRYPT_10819] [SWS_CRYPT_21617] [SWS_CRYPT_24800] [SWS_CRYPT_24801] [SWS_CRYPT_24811]
[RS_CRYPTO_02009]	The Crypto stack shall support separation of applications access rights to owner and users for each cryptographic object slot	[SWS_CRYPT_30117] [SWS_CRYPT_30121] [SWS_CRYPT_30122] [SWS_CRYPT_30300] [SWS_CRYPT_30400]
[RS_CRYPTO_02101]	The Crypto Stack shall provide interfaces to generate cryptographic keys for all supported primitives	[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_10014] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21212] [SWS_CRYPT_21213] [SWS_CRYPT_21412] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21614] [SWS_CRYPT_21615] [SWS_CRYPT_21813] [SWS_CRYPT_21814]
[RS_CRYPTO_02103]	The Crypto Stack shall support primitives to derive cryptographic key material from a base key material	[SWS_CRYPT_20748] [SWS_CRYPT_20749] [SWS_CRYPT_21500] [SWS_CRYPT_21600]
[RS_CRYPTO_02104]	The Crypto Stack shall support a primitive to exchange cryptographic keys with another entity	[SWS_CRYPT_20743] [SWS_CRYPT_20752] [SWS_CRYPT_20753] [SWS_CRYPT_20758] [SWS_CRYPT_21300] [SWS_CRYPT_21400] [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_10700] [SWS_CRYPT_20728] [SWS_CRYPT_20729] [SWS_CRYPT_20730] [SWS_CRYPT_20731] [SWS_CRYPT_20732]

Requirement	Description	Satisfied by
[RS_CRYPTO_02106]	The Crypto Stack shall provide interfaces for secure processing of passwords	[SWS_CRYPT_20736] [SWS_CRYPT_20738] [SWS_CRYPT_22300] [SWS_CRYPT_22321] [SWS_CRYPT_22400]
[RS_CRYPTO_02107]	The Crypto Stack shall support the algorithm specification in any key generation or derivation request	[SWS_CRYPT_10014] [SWS_CRYPT_13000] [SWS_CRYPT_13001] [SWS_CRYPT_13002] [SWS_CRYPT_13003] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21614] [SWS_CRYPT_21615]
[RS_CRYPTO_02108]	The Crypto Stack shall provide interfaces for management and usage of algorithm-specific domain parameters	[SWS_CRYPT_20414] [SWS_CRYPT_20719] [SWS_CRYPT_20720] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_20900] [SWS_CRYPT_21412] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21614] [SWS_CRYPT_21615] [SWS_CRYPT_21813] [SWS_CRYPT_21814] [SWS_CRYPT_22511] [SWS_CRYPT_22611] [SWS_CRYPT_22811] [SWS_CRYPT_24211] [SWS_CRYPT_24212] [SWS_CRYPT_24213] [SWS_CRYPT_24411]
[RS_CRYPTO_02109]	The Crypto Stack shall support interfaces for a unified Machine-wide storage and retrieval of different crypto objects	[SWS_CRYPT_10017] [SWS_CRYPT_10514] [SWS_CRYPT_30099] [SWS_CRYPT_30100]

Requirement	Description	Satisfied by
[RS_CRYPTO_02110]	The Crypto Stack shall support the prototyping of applications' access rights and content restrictions of key slot resources	[SWS_CRYPT_10010] [SWS_CRYPT_10813] [SWS_CRYPT_10818] [SWS_CRYPT_30118] [SWS_CRYPT_30300] [SWS_CRYPT_30301] [SWS_CRYPT_30302] [SWS_CRYPT_30303] [SWS_CRYPT_30304] [SWS_CRYPT_30305] [SWS_CRYPT_30306] [SWS_CRYPT_30307] [SWS_CRYPT_30308] [SWS_CRYPT_30309] [SWS_CRYPT_30310] [SWS_CRYPT_30311] [SWS_CRYPT_30350] [SWS_CRYPT_30351]
[RS_CRYPTO_02111]	The Crypto Stack shall provide applications a possibility to define usage restrictions of any new generated or derived key	[SWS_CRYPT_10015] [SWS_CRYPT_13100] [SWS_CRYPT_13101] [SWS_CRYPT_13102] [SWS_CRYPT_13103] [SWS_CRYPT_13104] [SWS_CRYPT_13105] [SWS_CRYPT_13106] [SWS_CRYPT_13107] [SWS_CRYPT_13108] [SWS_CRYPT_13109] [SWS_CRYPT_13110] [SWS_CRYPT_13111] [SWS_CRYPT_13112] [SWS_CRYPT_13113] [SWS_CRYPT_13114] [SWS_CRYPT_13115] [SWS_CRYPT_13116] [SWS_CRYPT_13117] [SWS_CRYPT_13118] [SWS_CRYPT_13119] [SWS_CRYPT_13120] [SWS_CRYPT_13121] [SWS_CRYPT_13122]

Requirement	Description	Satisfied by
		[SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21614] [SWS_CRYPT_21615] [SWS_CRYPT_30500] [SWS_CRYPT_30501] [SWS_CRYPT_30502] [SWS_CRYPT_30503] [SWS_CRYPT_30504] [SWS_CRYPT_30505] [SWS_CRYPT_30506] [SWS_CRYPT_30507] [SWS_CRYPT_30508] [SWS_CRYPT_30509] [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_10711] [SWS_CRYPT_10712] [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_20513] [SWS_CRYPT_20719] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_20730] [SWS_CRYPT_20738] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21514] [SWS_CRYPT_21515] [SWS_CRYPT_21618]
[RS_CRYPTO_02114]	The Crypto Stack shall support the possibility to restrict the allowed usage of a key individually to each user application	[SWS_CRYPT_10010] [SWS_CRYPT_30122] [SWS_CRYPT_30400] [SWS_CRYPT_30401] [SWS_CRYPT_30402] [SWS_CRYPT_30450] [SWS_CRYPT_30451]

Requirement	Description	Satisfied by
[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_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21614] [SWS_CRYPT_21813] [SWS_CRYPT_21814] [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_30005] [SWS_CRYPT_30300]
[RS_CRYPTO_02201]	The Crypto Stack shall provide interfaces to use symmetric encryption and decryption primitives	[SWS_CRYPT_20200] [SWS_CRYPT_20742] [SWS_CRYPT_20744] [SWS_CRYPT_23600] [SWS_CRYPT_23700] [SWS_CRYPT_23900]
[RS_CRYPTO_02202]	The Crypto Stack shall provide interfaces to use asymmetric encryption and decryption primitives	[SWS_CRYPT_20200] [SWS_CRYPT_20750] [SWS_CRYPT_20751] [SWS_CRYPT_20754] [SWS_CRYPT_20755] [SWS_CRYPT_20800] [SWS_CRYPT_21000] [SWS_CRYPT_22200] [SWS_CRYPT_22600] [SWS_CRYPT_22800] [SWS_CRYPT_23200]
[RS_CRYPTO_02203]	The Crypto Stack shall provide interfaces to use message authentication code primitives	[SWS_CRYPT_20300] [SWS_CRYPT_20319] [SWS_CRYPT_20746] [SWS_CRYPT_22100] [SWS_CRYPT_23300]
[RS_CRYPTO_02204]	The Crypto Stack shall provide interfaces to use digital signature primitives	[SWS_CRYPT_20319] [SWS_CRYPT_20754] [SWS_CRYPT_20755] [SWS_CRYPT_20756] [SWS_CRYPT_20757] [SWS_CRYPT_22200] [SWS_CRYPT_23200] [SWS_CRYPT_23300] [SWS_CRYPT_23400] [SWS_CRYPT_23500] [SWS_CRYPT_24100]
[RS_CRYPTO_02205]	The Crypto Stack shall provide interfaces to use hashing primitives	[SWS_CRYPT_20300] [SWS_CRYPT_20747] [SWS_CRYPT_21100] [SWS_CRYPT_23300]

Requirement	Description	Satisfied by
[RS_CRYPTO_02206]	The Crypto Stack shall provide interfaces to configure and use random number generation	[SWS_CRYPT_20739] [SWS_CRYPT_20740] [SWS_CRYPT_20741] [SWS_CRYPT_22900]
[RS_CRYPTO_02207]	The Crypto Stack shall provide interfaces to use authenticated symmetric encryption and decryption primitives	[SWS_CRYPT_20100] [SWS_CRYPT_20745]
[RS_CRYPTO_02208]	The Crypto Stack shall provide interfaces to use symmetric key wrapping primitives	[SWS_CRYPT_20743] [SWS_CRYPT_24000]
[RS_CRYPTO_02209]	The Crypto Stack shall provide interfaces to use asymmetric key encapsulation primitives	[SWS_CRYPT_20752] [SWS_CRYPT_20753] [SWS_CRYPT_21400] [SWS_CRYPT_21800] [SWS_CRYPT_21900]
[RS_CRYPTO_02302]	The Crypto Stack API shall support a streaming approach	[SWS_CRYPT_20312] [SWS_CRYPT_20313] [SWS_CRYPT_20314] [SWS_CRYPT_20315] [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_23711] [SWS_CRYPT_23712] [SWS_CRYPT_24700] [SWS_CRYPT_24714] [SWS_CRYPT_24715]
[RS_CRYPTO_02304]	The Crypto Stack API should support the possibility to move a state of a “counter mode” stream cipher to a random position	[SWS_CRYPT_23613]
[RS_CRYPTO_02305]	The Crypto Stack design shall separate cryptographic API from key access API	[SWS_CRYPT_20700] [SWS_CRYPT_30100]

Requirement	Description	Satisfied by
[RS_CRYPTO_02306]	The Crypto Stack shall support integration with a Public Key Infrastructure (PKI)	[SWS_CRYPT_20759] [SWS_CRYPT_24212] [SWS_CRYPT_24311] [SWS_CRYPT_24313] [SWS_CRYPT_24314] [SWS_CRYPT_24411] [SWS_CRYPT_24511] [SWS_CRYPT_24611] [SWS_CRYPT_40001] [SWS_CRYPT_40002] [SWS_CRYPT_40099] [SWS_CRYPT_40100] [SWS_CRYPT_40200] [SWS_CRYPT_40203] [SWS_CRYPT_40300] [SWS_CRYPT_40400] [SWS_CRYPT_40500] [SWS_CRYPT_40600] [SWS_CRYPT_40700] [SWS_CRYPT_40800]
[RS_CRYPTO_02307]	The Crypto Stack design shall separate cryptographic API from the PKI API	[SWS_CRYPT_20700] [SWS_CRYPT_20759] [SWS_CRYPT_24200] [SWS_CRYPT_24300] [SWS_CRYPT_24400] [SWS_CRYPT_24500] [SWS_CRYPT_24600]
[RS_CRYPTO_02308]	The Crypto Stack shall support a unified cryptographic primitives naming convention, common for all suppliers	[SWS_CRYPT_20611] [SWS_CRYPT_20711] [SWS_CRYPT_20712] [SWS_CRYPT_20734]
[RS_CRYPTO_02309]	The Crypto Stack API shall support the run-time configurable usage style	[SWS_CRYPT_20110] [SWS_CRYPT_20211] [SWS_CRYPT_20212] [SWS_CRYPT_20213] [SWS_CRYPT_20214] [SWS_CRYPT_20215] [SWS_CRYPT_20311] [SWS_CRYPT_20411] [SWS_CRYPT_20412] [SWS_CRYPT_20516] [SWS_CRYPT_20612] [SWS_CRYPT_20613] [SWS_CRYPT_20734] [SWS_CRYPT_20911] [SWS_CRYPT_20912] [SWS_CRYPT_20913] [SWS_CRYPT_20914] [SWS_CRYPT_20915] [SWS_CRYPT_20916] [SWS_CRYPT_20919] [SWS_CRYPT_21212] [SWS_CRYPT_21213] [SWS_CRYPT_21214] [SWS_CRYPT_21215]

Requirement	Description	Satisfied by
		[SWS_CRYPT_21511] [SWS_CRYPT_21711] [SWS_CRYPT_21712] [SWS_CRYPT_21713] [SWS_CRYPT_21714] [SWS_CRYPT_21911] [SWS_CRYPT_21912] [SWS_CRYPT_22011] [SWS_CRYPT_22411] [SWS_CRYPT_22412] [SWS_CRYPT_23411] [SWS_CRYPT_23412] [SWS_CRYPT_23413] [SWS_CRYPT_23611] [SWS_CRYPT_23612] [SWS_CRYPT_23711] [SWS_CRYPT_23912] [SWS_CRYPT_24711] [SWS_CRYPT_24712] [SWS_CRYPT_24713] [SWS_CRYPT_24716]
[RS_CRYPTO_02310]	The Crypto Stack API shall support an efficient mechanism of error states notification	[SWS_CRYPT_10099] [SWS_CRYPT_16000] [SWS_CRYPT_16100] [SWS_CRYPT_16200] [SWS_CRYPT_16300] [SWS_CRYPT_16400] [SWS_CRYPT_16411] [SWS_CRYPT_16500] [SWS_CRYPT_16600] [SWS_CRYPT_16700] [SWS_CRYPT_16800] [SWS_CRYPT_16811] [SWS_CRYPT_16900] [SWS_CRYPT_16911] [SWS_CRYPT_16912] [SWS_CRYPT_17000] [SWS_CRYPT_17011] [SWS_CRYPT_17100] [SWS_CRYPT_19900] [SWS_CRYPT_19950] [SWS_CRYPT_19951]

Requirement	Description	Satisfied by
[RS_CRYPTO_02311]	The Crypto Stack API specification should be complete and allow flexible usage of the stack functionality	[SWS_CRYPT_10030] [SWS_CRYPT_10031] [SWS_CRYPT_10032] [SWS_CRYPT_10033] [SWS_CRYPT_10150] [SWS_CRYPT_10151] [SWS_CRYPT_10152] [SWS_CRYPT_10153] [SWS_CRYPT_10154] [SWS_CRYPT_10155] [SWS_CRYPT_10210] [SWS_CRYPT_10311] [SWS_CRYPT_10312] [SWS_CRYPT_10451] [SWS_CRYPT_10452] [SWS_CRYPT_10453] [SWS_CRYPT_10454] [SWS_CRYPT_10455] [SWS_CRYPT_10456] [SWS_CRYPT_10500] [SWS_CRYPT_10510] [SWS_CRYPT_10512] [SWS_CRYPT_10550] [SWS_CRYPT_10551] [SWS_CRYPT_10552] [SWS_CRYPT_10553] [SWS_CRYPT_10554] [SWS_CRYPT_10555] [SWS_CRYPT_10600] [SWS_CRYPT_10601] [SWS_CRYPT_10602] [SWS_CRYPT_10603] [SWS_CRYPT_10604] [SWS_CRYPT_10605] [SWS_CRYPT_10701] [SWS_CRYPT_10710] [SWS_CRYPT_10750] [SWS_CRYPT_10751] [SWS_CRYPT_10752] [SWS_CRYPT_10753] [SWS_CRYPT_10810] [SWS_CRYPT_10812] [SWS_CRYPT_10814] [SWS_CRYPT_10815] [SWS_CRYPT_10816] [SWS_CRYPT_10817] [SWS_CRYPT_20001] [SWS_CRYPT_20002]

Requirement	Description	Satisfied by
		[SWS_CRYPT_20003] [SWS_CRYPT_20004] [SWS_CRYPT_20005] [SWS_CRYPT_20210] [SWS_CRYPT_20216] [SWS_CRYPT_20217] [SWS_CRYPT_20400] [SWS_CRYPT_20500] [SWS_CRYPT_20503] [SWS_CRYPT_20511] [SWS_CRYPT_20518] [SWS_CRYPT_20600] [SWS_CRYPT_20601] [SWS_CRYPT_20602] [SWS_CRYPT_20701] [SWS_CRYPT_20702] [SWS_CRYPT_20710] [SWS_CRYPT_20901] [SWS_CRYPT_20902] [SWS_CRYPT_20903] [SWS_CRYPT_20904] [SWS_CRYPT_20917] [SWS_CRYPT_20918] [SWS_CRYPT_21611] [SWS_CRYPT_21612] [SWS_CRYPT_21613] [SWS_CRYPT_21616] [SWS_CRYPT_21700] [SWS_CRYPT_21910] [SWS_CRYPT_22000] [SWS_CRYPT_22010] [SWS_CRYPT_22311] [SWS_CRYPT_22312] [SWS_CRYPT_22313] [SWS_CRYPT_22314] [SWS_CRYPT_22315] [SWS_CRYPT_22316] [SWS_CRYPT_22317] [SWS_CRYPT_22318] [SWS_CRYPT_22320] [SWS_CRYPT_22512] [SWS_CRYPT_22711] [SWS_CRYPT_22712] [SWS_CRYPT_22713] [SWS_CRYPT_22714] [SWS_CRYPT_22901] [SWS_CRYPT_22911] [SWS_CRYPT_22913]

Requirement	Description	Satisfied by
		[SWS_CRYPT_23011] [SWS_CRYPT_23012] [SWS_CRYPT_23013] [SWS_CRYPT_23014] [SWS_CRYPT_23015] [SWS_CRYPT_23016] [SWS_CRYPT_23310] [SWS_CRYPT_23311] [SWS_CRYPT_23410] [SWS_CRYPT_23811] [SWS_CRYPT_24011] [SWS_CRYPT_24012] [SWS_CRYPT_24013] [SWS_CRYPT_24014] [SWS_CRYPT_24111] [SWS_CRYPT_24112] [SWS_CRYPT_24312] [SWS_CRYPT_24412] [SWS_CRYPT_24710] [SWS_CRYPT_30002] [SWS_CRYPT_30101] [SWS_CRYPT_30104] [SWS_CRYPT_30110] [SWS_CRYPT_30113] [SWS_CRYPT_30114] [SWS_CRYPT_30115] [SWS_CRYPT_30116] [SWS_CRYPT_30119] [SWS_CRYPT_30201] [SWS_CRYPT_30210] [SWS_CRYPT_30350] [SWS_CRYPT_30351] [SWS_CRYPT_30450] [SWS_CRYPT_30451] [SWS_CRYPT_30550] [SWS_CRYPT_30551] [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]

Requirement	Description	Satisfied by
		[SWS_CRYPT_40156] [SWS_CRYPT_40157] [SWS_CRYPT_40158] [SWS_CRYPT_40159] [SWS_CRYPT_40211] [SWS_CRYPT_40212] [SWS_CRYPT_40213] [SWS_CRYPT_40214] [SWS_CRYPT_40215] [SWS_CRYPT_40216] [SWS_CRYPT_40217] [SWS_CRYPT_40218] [SWS_CRYPT_40219] [SWS_CRYPT_40220] [SWS_CRYPT_40221] [SWS_CRYPT_40311] [SWS_CRYPT_40312] [SWS_CRYPT_40403] [SWS_CRYPT_40411] [SWS_CRYPT_40412] [SWS_CRYPT_40413] [SWS_CRYPT_40414] [SWS_CRYPT_40415] [SWS_CRYPT_40416] [SWS_CRYPT_40511] [SWS_CRYPT_40601] [SWS_CRYPT_40602] [SWS_CRYPT_40603] [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_40623] [SWS_CRYPT_40624] [SWS_CRYPT_40625] [SWS_CRYPT_40626] [SWS_CRYPT_40627] [SWS_CRYPT_40628] [SWS_CRYPT_40629] [SWS_CRYPT_40630]

Requirement	Description	Satisfied by
		[SWS_CRYPT_40631] [SWS_CRYPT_40632] [SWS_CRYPT_40633] [SWS_CRYPT_40634] [SWS_CRYPT_40635] [SWS_CRYPT_40701] [SWS_CRYPT_40702] [SWS_CRYPT_40711] [SWS_CRYPT_40801] [SWS_CRYPT_40802] [SWS_CRYPT_40811]
[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_10011] [SWS_CRYPT_10012] [SWS_CRYPT_10017] [SWS_CRYPT_10513] [SWS_CRYPT_10514] [SWS_CRYPT_20099] [SWS_CRYPT_20614] [SWS_CRYPT_20700] [SWS_CRYPT_30099] [SWS_CRYPT_30100] [SWS_CRYPT_30120] [SWS_CRYPT_30129] [SWS_CRYPT_30130] [SWS_CRYPT_30131]
[RS_CRYPTO_02403]	The Crypto Stack shall support isolating keys and requests	[SWS_CRYPT_21200] [SWS_CRYPT_22500] [SWS_CRYPT_22700] [SWS_CRYPT_23800]
[RS_CRYPTO_02404]	Design of the Crypto Stack interfaces shall support minimization of resources consumption	[SWS_CRYPT_10001] [SWS_CRYPT_10002] [SWS_CRYPT_10010] [SWS_CRYPT_10011] [SWS_CRYPT_10012] [SWS_CRYPT_10013] [SWS_CRYPT_10014] [SWS_CRYPT_10015] [SWS_CRYPT_10100] [SWS_CRYPT_10150] [SWS_CRYPT_10151] [SWS_CRYPT_10152] [SWS_CRYPT_10153] [SWS_CRYPT_10154] [SWS_CRYPT_10155] [SWS_CRYPT_10200] [SWS_CRYPT_10211] [SWS_CRYPT_10300] [SWS_CRYPT_10400] [SWS_CRYPT_10411] [SWS_CRYPT_10450] [SWS_CRYPT_10451] [SWS_CRYPT_10452] [SWS_CRYPT_10453]

Requirement	Description	Satisfied by
		[SWS_CRYPT_10454] [SWS_CRYPT_10455] [SWS_CRYPT_10456] [SWS_CRYPT_10511] [SWS_CRYPT_10513] [SWS_CRYPT_10515] [SWS_CRYPT_10550] [SWS_CRYPT_10551] [SWS_CRYPT_10552] [SWS_CRYPT_10553] [SWS_CRYPT_10554] [SWS_CRYPT_10555] [SWS_CRYPT_10600] [SWS_CRYPT_10601] [SWS_CRYPT_10602] [SWS_CRYPT_10603] [SWS_CRYPT_10604] [SWS_CRYPT_10605] [SWS_CRYPT_10611] [SWS_CRYPT_10612] [SWS_CRYPT_10801] [SWS_CRYPT_10802] [SWS_CRYPT_20101] [SWS_CRYPT_20315] [SWS_CRYPT_20316] [SWS_CRYPT_20317] [SWS_CRYPT_20318] [SWS_CRYPT_20501] [SWS_CRYPT_20502] [SWS_CRYPT_20703] [SWS_CRYPT_20704] [SWS_CRYPT_20705] [SWS_CRYPT_20706] [SWS_CRYPT_20707] [SWS_CRYPT_20713] [SWS_CRYPT_20714] [SWS_CRYPT_20715] [SWS_CRYPT_20716] [SWS_CRYPT_20719] [SWS_CRYPT_20720] [SWS_CRYPT_20721] [SWS_CRYPT_20722] [SWS_CRYPT_20723] [SWS_CRYPT_20724] [SWS_CRYPT_20725] [SWS_CRYPT_20726] [SWS_CRYPT_20727] [SWS_CRYPT_20733]

Requirement	Description	Satisfied by
		[SWS_CRYPT_20736] [SWS_CRYPT_20738] [SWS_CRYPT_20739] [SWS_CRYPT_20740] [SWS_CRYPT_20741] [SWS_CRYPT_20742] [SWS_CRYPT_20743] [SWS_CRYPT_20744] [SWS_CRYPT_20745] [SWS_CRYPT_20746] [SWS_CRYPT_20747] [SWS_CRYPT_20748] [SWS_CRYPT_20749] [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_20759] [SWS_CRYPT_20760] [SWS_CRYPT_20801] [SWS_CRYPT_21001] [SWS_CRYPT_21101] [SWS_CRYPT_21201] [SWS_CRYPT_21301] [SWS_CRYPT_21311] [SWS_CRYPT_21312] [SWS_CRYPT_21400] [SWS_CRYPT_21401] [SWS_CRYPT_21411] [SWS_CRYPT_21412] [SWS_CRYPT_21501] [SWS_CRYPT_21512] [SWS_CRYPT_21513] [SWS_CRYPT_21514] [SWS_CRYPT_21515] [SWS_CRYPT_21601] [SWS_CRYPT_21618] [SWS_CRYPT_21800] [SWS_CRYPT_21801] [SWS_CRYPT_21811] [SWS_CRYPT_21812] [SWS_CRYPT_21813] [SWS_CRYPT_21814]

Requirement	Description	Satisfied by
		[SWS_CRYPT_22101] [SWS_CRYPT_22201] [SWS_CRYPT_22301] [SWS_CRYPT_22319] [SWS_CRYPT_22401] [SWS_CRYPT_22501] [SWS_CRYPT_22511] [SWS_CRYPT_22512] [SWS_CRYPT_22701] [SWS_CRYPT_22714] [SWS_CRYPT_23001] [SWS_CRYPT_23002] [SWS_CRYPT_23201] [SWS_CRYPT_23301] [SWS_CRYPT_23501] [SWS_CRYPT_23511] [SWS_CRYPT_23512] [SWS_CRYPT_23513] [SWS_CRYPT_23601] [SWS_CRYPT_23701] [SWS_CRYPT_23801] [SWS_CRYPT_23811] [SWS_CRYPT_24000] [SWS_CRYPT_24001] [SWS_CRYPT_24015] [SWS_CRYPT_24016] [SWS_CRYPT_24017] [SWS_CRYPT_24101] [SWS_CRYPT_24201] [SWS_CRYPT_24212] [SWS_CRYPT_24301] [SWS_CRYPT_24401] [SWS_CRYPT_24411] [SWS_CRYPT_24501] [SWS_CRYPT_24511] [SWS_CRYPT_24601] [SWS_CRYPT_24611] [SWS_CRYPT_30001] [SWS_CRYPT_30103] [SWS_CRYPT_30300] [SWS_CRYPT_30400] [SWS_CRYPT_30500] [SWS_CRYPT_40201] [SWS_CRYPT_40202] [SWS_CRYPT_40301] [SWS_CRYPT_40302] [SWS_CRYPT_40401] [SWS_CRYPT_40402] [SWS_CRYPT_40501]
[RS_CRYPTO_02405]	The Crypto Stack shall support the key slots identification in a way independent from a concrete deployment	[SWS_CRYPT_10013] [SWS_CRYPT_30103] [SWS_CRYPT_30111] [SWS_CRYPT_30127] [SWS_CRYPT_30128] [SWS_CRYPT_30132]

Requirement	Description	Satisfied by
[RS_CRYPTO_02406]	The Crypto Stack shall support its efficient usage in the “real-time” and “non-real-time” modes	[SWS_CRYPT_20717] [SWS_CRYPT_20718]

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 Security Management (SEC) for AUTOSAR Adaptive is such a functional cluster and is part of “AUTOSAR Runtime for Adaptive Applications” - ARA. The functional cluster consists of multiple modules. The Crypto Stack is a module of this functional cluster that offers interfaces to Adaptive applications. It is responsible for the construction and supervision of cryptographic primitives.

The Crypto Stack provides the infrastructure to access multiple implementations of cryptographic algorithms through a standardized interface.

This specification includes the syntax of the API, the relationship of the API to the model and describes semantics. The specification does not pose constraints on the internal architecture and implementation of the Crypto Stack.

7.1 Architectural concepts

The Crypto Stack of AUTOSAR Adaptive can be logically divided into the following parts:

- Language binding
- Drivers
- Crypto Stack management software.

There are several types of interfaces available in the context of the Crypto Stack:

- **Public Interface**
  Part of the AUTOSAR Adaptive API and specified in this document. This is the standardized API presented in the namespace `ara::crypto`. It includes a few sub-domain APIs presented by following namespaces: `crypt, keys, x509`.
- **Protected Interface**
  Used for interaction between functional clusters. This may be a custom API, but it can also re-use the Public Interface.
- **Private Interface**
  Used for interaction within the module. These interfaces are not described in the specification and are implementation-specific.

For the design of the ARA Crypto API 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.

Therefore the API of the Crypto Stack follows a specific set of design decisions:

- It uses a pure virtual API to access different algorithms through a unified interface.
- The memory management controllable by the caller.
- Its API has zero-copy capabilities.
- A "comfort layer" provides functionality like asynchronous operation (not implemented yet).

7.2 Integration of Adaptive Application and Crypto Stack

The `Adaptive Application` should not have direct access to keys within its own process. Therefore the Crypto Stack has to support features for isolating Adaptive Applications from the Crypto Stack implementation. The following separation mechanisms are envisioned:

1. Process isolation
2. Hardware isolation

The two mechanisms will be outlined briefly below.

7.2.1 Process isolation

The implementer/integrator of this specification may choose to isolate the cryptographic algorithm implementation from the `Adaptive Application`s by means of separating them into two different processes. Generally it is strongly recommended to separate the Crypto Stack on two domains: "front-end" and isolated "back-end". At least the Crypto Stack back-end shall include all functionality that is subject of access control and implementations of cryptographic transformations that handle secret/private key material. It should be done in following ways:

The Crypto Stack back-end can be implemented in form of privileged independent process (e.g. driver or daemon). But independently from details of the Crypto Stack back-end implementation, its front-end should be presented to `Adaptive Application`'s developers in form of a library, which hides all implementation and communication details. The front-end library can directly implement key-less transformations (like hashing) and public key transformations (like signature verification). The Crypto Stack interface visible to the `Adaptive Application` developers is specified in chapter 8.

7.2.2 Hardware isolation

The implementer/integrator of this specification may choose to isolate the cryptographic algorithm implementation from the `Adaptive Application`s by means of separating them using a hardware mechanism (e.g. HSM, TPM). It should be done in following ways:

All communications with hardware should be implemented in the Crypto Stack "back-end" (in form of a driver), all other aspects are similar to the section 7.2.1. Switching between a hardware and a software based implementations is fully transparent for `Adaptive Application` developers, in both cases they use the API specified in chapter 8.

7.3 Supported algorithms

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

- Random Number Generation
  - Deterministic Random Number Generator (DRNG)
  - True Random Number Generator (TRNG)
- Symmetric Ciphers
  - AES
    - Key Length: 128, 192 and 256 bits
    - Modes: ECB, OFB, CFB, CBC, CTR, GCM, CCM, Poly1305
  - Camellia
    - Key Length: 128, 192 and 256 bits
    - Modes: ECB, OFB, CFB, CBC, CTR, GCM, CCM
  - ChaCha20
- Asymmetric Encryption/Decryption and Signature Handling
  - RSA
    - Key Length: 2048, 3072 and 4096 bits
    - Padding: PKCS#1 v2.2
  - Curve25519/Ed25519
  - NIST curves P256, P384 and P521 / ECDSA
- Hash Functions

  - SHA-1
  - SHA-2
    * Length: 256, 384 and 512 bits
  - SHA-3
    * Length: 256, 384 and 512 bits
- Message Authentication Code (MAC)
  - CBC-MAC
  - CMAC
  - GMAC
  - HMAC
- Key Agreement
  - Diffie-Hellman (DH)
  - Elliptic Curve DH (ECDH)
- Key Derivation Function
  - HKDF
  - PBKDF1
  - PBKDF2
- Key Encapsulation Mechanism
  - RSA-KEM
  - ECIES-KEM
  - PSEC-KEM
  - ACE-KEM

The Crypto Stack should support handling the following cryptographic objects:

- Certificate Management
  - Handling of X.509 Certificates
  - Import/Export in DER and PEM formats
  - Creation of CSRs

7.4 Crypto API structure

Crypto API provided by Crypto Stack to consumers is presented by 4 different Provider types, each of them implements specific domain of cryptography-related functionality:

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

2. **Key Storage Provider** (KSP, namespace `ara::crypto::keys`) is responsible for secure (confidential and/or authentic) storage of different type key material (public/private/secret keys, seeds, domain parameters) and other security critical cryptographic objects (digital signatures, hash, MAC/HMAC tags). Crypto API 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 crypto objects. All back-end KSP are hidden from the consumers (under public Crypto API). KSP implementation (similar to CP) 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). Crypto Stack supports only single instance of the CMP and it is completely independent from CP and KSP implementation details, therefore CMP and CP/KSP may be provided by completely independent suppliers. **Note:** CMP works with non-confidential objects only.

4. **Objects Coding Provider** (OCP, planned but not implemented yet namespace `ara::crypto::code`) is responsible for encoding/decoding of different crypto objects to/from well-known standardized formats. Crypto Stack plans support of only single instance of the OCP and it should be completely independent from any details of CP/KSP implementation. But OCP may provide internal standardized interfaces for all CPs of CryptoStack.

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

7.5 Cryptographic Primitives Naming Convention

Future Crypto Providers can support some crypto algorithms that are not well-known/standardized today, therefore this API specification 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. I.e. a concrete set of crypto algorithms supported by Crypto Provider remain at the discretion of a supplier.

Any name of a crypto algorithm should satisfy to following rules

1. Only latin alphanumeric characters and 6 defined below delimiters can be used for crypto algorithm definition.
2. Case of letters does not matter, i.e. all comparisons of the identifiers must be always case-insensitive.
3. Any name of a crypto algorithm should satisfy to following structure:

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

   where

   - "`{TargetTransformation(Mode)}`" – a specificator of target transformation: for complex transformations it is a mode name, but for fully-defined algorithms it is just their name;
   - "`{SupportingAlgorithms}`" – a specificator of basic crypto algorithm(s) including key length and/or block length;
   - "`{Encoding&Padding}`" – a specificator 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 doesn't add anything, but if they are not aligned then applies a padding by '`\0`' bytes up to the block boundary.
4. 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 specificators that define key-length or block-length in bits.
   - '`+`' – separator between a few base algorithms' specifications for a cascade transformation definition.

- `','` – separator between a few base algorithms' specifications for a case if the whole algorithm is based on a few types of basic transformations.
- `'.'` – separator between a common name of a standard and its specific part or its version that precises a specification of concrete transformation.

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

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

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

Examples of fully defined transformations:

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

5. If an algorithm support a few variable length parameters then they should be specified in following order: key, I/O-block or output digest, IV or input block (e.g.: `"Kalyna-512-256"` means block cipher Kalina with 512-bit key and 256-bit block).
6. If a transformation is based on a few basic cryptoalgorithms then they should be specified in an order corresponding to the level of their application (see example below for RSA).
7. Following Mode specificators can be used for RSA-based algorithms:
   - `"SIG"` – signature primitive (e.g.: `"SIG/RSA-2048,SHA-160/PKCS1.v1_5,EMSA"`)
   - `"VER"` – verification primitive (e.g.: `"VER/RSA-2048,SHA-160/PKCS1.v1_5,EMSA"`)

  - "ENC" – encryption primitive (e.g.: "ENC/RSA-2048,MGF1,SHA-160/PKCS1.v1_5,EME", "ENC/RSA-4096,MGF1,SHA2-256/OAEP,EME")
  - "DEC" – decryption primitive (e.g.: "DEC/RSA-2048,MGF1,SHA-160/PKCS1.v1_5,EME", "DEC/RSA-4096,MGF1,SHA2-256/OAEP,EME")
  - "KEM" – Key Encapsulation Mechanism (e.g.: "KEM/RSA-2048,AES-128,KDF3,SHA-256")

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

7.6 Primitives Usage Examples

General usage sequence of any crypto primitive interface:

1. Optional creation of a Crypto Context (if there is no an existing one that can be reused)
2. Initialization of a Crypto Context (loading key material, initialization vector (IV) and other parameters expected by the transformation)
3. (Iterative) Data processing (the data flow may be presented by a sequence of data chunks)
4. Finalization of data processing (optional for some algorithms)

Code snippets presented below demonstrate usage of the following cryptographic primitives:

- Encryption/Decryption by Stream Cipher and Authenticated Stream Cipher
- Hash-function calculation/verification
- Message Authentication Code calculation/verification
- Digital Signature calculation/verification

Note: Presented below code snippets are simplified and dedicated only for demonstration of the correct call sequences! All errors processing logic is omitted, therefore they cannot be used "as is"!

7.6.1 Factory routines

```
/// @brief Factory that creates stream cipher context according to directly
    provided algorithm name.
/// @param[in] algName  Name of the target crypto algorithm
/// @return Unique smart pointer to the created context instance
/// @note @c algName must be one of known names of crypto primitives
    defined in the Autosar specification!
StreamCipherCtx::Uptr createCipher(StringView algName)
{
    CryptoProvider::Sptr cp = LoadCryptoProvider();
    CryptoAlgId algId = cp->ConvertToAlgId(algName);
    return cp->CreateStreamCipherCtx(algId);
}
```

Listing 7.1: Stream Cipher context creation by an algorithm name

```
/// @brief Factory that creates stream cipher context according to
    algorithm specification in the provided symmetric key object.
/// @param[in] key  Symmetric key object
/// @return Unique smart pointer to the created context instance
/// @note Key object @b must have complete specification of the target
    algorithm!
StreamCipherCtx::Uptr createCipher(const SymmetricKey & key)
{
    CryptoProvider & cp = key.MyProvider();
    CryptoAlgId algId = key.GetPrimitiveId();  ///< Get algorithm ID from
    the key object
    return cp.CreateStreamCipherCtx(algId);
}
```

Listing 7.2: Stream Cipher context creation by a key object

```
/// @brief Factory that creates authenticated stream cipher context
    according to directly provided algorithm name.
/// @param[in] algName  Name of the target crypto algorithm
/// @return Unique smart pointer to the created context instance
/// @note @c algName must be one of known names of crypto primitives
    defined in the Autosar specification!
AuthnStreamCipherCtx::Uptr createAuthCipher(StringView algName)
{
    CryptoProvider::Sptr cp = LoadCryptoProvider();
    CryptoAlgId algId = cp->ConvertToAlgId(algName);
    return cp->CreateAuthnStreamCipherCtx(algId);
}
```

Listing 7.3: Authenticated Stream Cipher context creation by an algorithm name

```
/// @brief Factory that creates authenticated stream cipher context
    according to algorithm specification in the provided symmetric key
    object.
/// @param[in] algName  Name of the target crypto algorithm
/// @return Unique smart pointer to the created context instance
/// @note @c algName must be one of known names of crypto primitives
    defined in the Autosar specification!
AuthnStreamCipherCtx::Uptr createAuthCipher(const SymmetricKey & key)
{
    CryptoProvider & cp = key.MyProvider();
    CryptoAlgId algId = key.GetPrimitiveId();  ///< Get algorithm ID from
    the key object
    return cp.CreateAuthnStreamCipherCtx(algId);
}
```

Listing 7.4: Authenticated Stream Cipher context creation by by a key object

```
/// @brief Factory that creates hash function context according to directly
     provided algorithm name.
/// @param[in] algName  Name of the target crypto algorithm
/// @return Unique smart pointer to the created context instance
/// @note @c algName must be one of known names of crypto primitives
    defined in the Autosar specification!
HashFunctionCtx::Uptr createHashFunc(StringView algName)
{
    CryptoProvider::Sptr cp = LoadCryptoProvider();
    return cp->CreateHashFunctionCtx(cp->ConvertToAlgId(algName));
}
```

Listing 7.5: Hash Function context creation by an algorithm name

```
/// @brief Factory that creates hash function context according to a
    specification provided in the signature object.
/// @param[in] sig  Signature object
/// @return Unique smart pointer to the created context instance
HashFunctionCtx::Uptr createHashFunc(const Signature & sig,
    DomainParameters::Sptrc params = nullptr)
{
    return sig.MyProvider().CreateHashFunctionCtx(sig.GetHashAlgId());
}
```

Listing 7.6: Hash Function context creation by a signature object

```
/// @brief Factory that creates Message Authentiaction Code (MAC) context
    according to directly provided algorithm name.
/// @param[in] algName  Name of the target crypto algorithm
/// @return Unique smart pointer to the created context instance
/// @note @c algName must be one of known names of crypto primitives
    defined in the Autosar specification!
MessageAuthnCodeCtx::Uptr createMac(StringView algName)
{
    CryptoProvider::Sptr cp = LoadCryptoProvider();
    return cp->CreateMessageAuthnCodeCtx(cp->ConvertToAlgId(algName));
}
```

Listing 7.7: Message Authentiaction Code context creation by an algorithm name

```
/// @brief Factory that creates a signer private context according to
    directly provided algorithm name.
/// @param[in] algName  Name of the target crypto algorithm
/// @return Unique smart pointer to the created context instance
/// @note @c algName must be one of known names of crypto primitives
    defined in the Autosar specification!
SignerPrivateCtx::Uptr createSigner(StringView algName)
{
    CryptoProvider::Sptr cp = LoadCryptoProvider();
    return cp->CreateSignerPrivateCtx(cp->ConvertToAlgId(algName));
}
```

Listing 7.8: Signer private-key context creation by an algorithm name

```
/// @brief Factory that creates a digital signature verifier context
    according to directly provided algorithm name.
/// @param[in] algName  Name of the target crypto algorithm
/// @return Unique smart pointer to the created context instance
/// @note @c algName must be one of known names of crypto primitives
    defined in the Autosar specification!
VerifierPublicCtx::Uptr createVerifier(StringView algName)
{
    CryptoProvider::Sptr cp = LoadCryptoProvider();
    return cp->CreateVerifierPublicCtx(cp->ConvertToAlgId(algName));
}
```

Listing 7.9: Digital signature verifier context creation by an algorithm name

```
/// @brief Factory that creates a digital signature verifier context
    according to algorithm specified in the signature object.
/// @param[in] sig  Signature object
/// @return Unique smart pointer to the created context instance
VerifierPublicCtx::Uptr createVerifier(const Signature & sig)
{
    return sig.MyProvider().CreateVerifierPublicCtx(sig.GetPrimitiveId());
}
```

Listing 7.10: Digital signature verifier context creation by a signature object

7.6.2 Initialization routiness

```
/// @brief Initialize stream cipher context for encryption or decryption.
/// @param[in] cipherCtx  Target stream cipher context for the
    initialization
/// @param[in] key  Symmetric key that should be loaded to the target
    context
/// @param[in] iv  Initialization Vector (IV) that should be loaded to the
    target context (object content hidden from the application)
/// @param[in] directTransform  Flag that controls direction of the
    transformation: @c true - encrypt, @c false - decrypt
/// @param[in] params  Optional pointer to a "domain parameters" object (
    cipher-specific extra configuration)
void initCipher(StreamCipherCtx & cipherCtx, const SymmetricKey & key,
    const SecretSeed & iv
    , bool directTransform = true, DomainParameters::Sptrc params = nullptr
    )
{
    cipherCtx.Reset(params);  ///< Optional call
    cipherCtx.SetKey(key, directTransform);  ///< Mandatory call
    cipherCtx.Start(iv);  ///< Mandatory call
}
```

Listing 7.11: Stream Cipher context initialization (variant 1)

```
/// @brief Initialize stream cipher context for encryption or decryption.
/// @note Similar to the previous one, but the application can directly
    control the IV value.
/// @param[in] cipherCtx  Target stream cipher context for the
    initialization
/// @param[in] key  Symmetric key that should be loaded to the target
    context
```

```
/// @param[in] iv  Initialization Vector that should be loaded to the
    target context (value is default or supplied by the application)
/// @param[in] directTransform  Flag that controls direction of the
    transformation: @c true - encrypt, @c false - decrypt
/// @param[in] params  Optional pointer to a "domain parameters" object (
    cipher-specific extra configuration)
void initCipher(StreamCipherCtx & cipherCtx, const SymmetricKey & key,
    ReadOnlyMemRegion iv = ReadOnlyMemRegion()
    , bool directTransform = true, DomainParameters::Sptrc params = nullptr
    )
{
    cipherCtx.Reset(params);  ///< Optional call
    cipherCtx.SetKey(key, directTransform);  ///< Mandatory call
    cipherCtx.Start(iv);  ///< Mandatory call
}
```

Listing 7.12: Stream Cipher context initialization (variant 2)

```
/// @brief Initialize Message Authentication Code (MAC) context.
/// @param[in] macCtx  Target MAC context for the initialization
/// @param[in] key  Symmetric key that should be loaded to the target
    context
/// @param[in] iv  Initialization Vector that should be loaded to the
    target context (value is default or supplied by the application)
/// @param[in] params  Optional pointer to a "domain parameters" object (
    MAC-specific extra configuration)
void initMac(MessageAuthnCodeCtx & macCtx, const SymmetricKey & key
    , ReadOnlyMemRegion iv = ReadOnlyMemRegion(), DomainParameters::Sptrc
    params = nullptr)
{
    macCtx.Reset(params);  ///< Optional call
    macCtx.SetKey(key);  ///< Mandatory call
    macCtx.Start(iv);  ///< Mandatory call
}
```

Listing 7.13: Message Authentication Code context initialization

```
/// @brief Initialize a hash function context.
/// @param[in] hashCtx  Target hash-function context
/// @param[in] iv  Optional initialization vector
/// @param[in] params  Optional pointer to a "domain parameters" object (
    Hash-specific extra configuration)
/// @return Unique smart pointer to the created context instance
void initHashFunc(HashFunctionCtx & hashCtx, ReadOnlyMemRegion iv =
    ReadOnlyMemRegion()
    , DomainParameters::Sptrc params = nullptr)
{
    hashCtx.Reset(params);  ///< Optional call
    hashCtx.Start(iv);  ///< Mandatory call
}
```

Listing 7.14: Hash Function context initialization

```
/// @brief Initialize a signer private context by provided private key
    object.
/// @param[in] signerCtx  Target digital signature calculation context
/// @param[in] key  Private key object
/// @param[in] params  Optional pointer to a domain parameters object
/// @return Unique smart pointer to the created context instance
void initSigner(SignerPrivateCtx & signerCtx, const PrivateKey & key,
    DomainParameters::Sptrc params = nullptr)
{
    signerCtx.Reset(params);  ///< Optional call
    signerCtx.SetKey(key);  ///< Mandatory call
}
```

Listing 7.15: Digital signature verifier context initialization

7.6.3 Factory and Initialization (combined) routines

```
/// @brief Factory that creates and initialize a signer private context
    according to algorithm specified in the private key object.
/// @param[in] key  Private key object
/// @param[in] params  Optional pointer to a domain parameters object
/// @return Unique smart pointer to the created context instance
SignerPrivateCtx::Uptr initSigner(const PrivateKey & key, DomainParameters
    ::Sptrc params = nullptr)
{
    SignerPrivateCtx::Uptr ctx = key.MyProvider().CreateSignerPrivateCtx(
    key.GetPrimitiveId());
    ctx->Reset(params);
    ctx->SetKey(key);
    return ctx;
}
```

Listing 7.16: Signer private-key context creation and initialization

```
/// @brief Factory that creates and initialize a digital signature verifier
     context according to algorithm specified in the public key object.
/// @param[in] key  Public key object
/// @param[in] params  Optional pointer to a domain parameters object
/// @return Unique smart pointer to the created context instance
VerifierPublicCtx::Uptr initVerifier(const PublicKey & key,
    DomainParameters::Sptrc params = nullptr)
{
    VerifierPublicCtx::Uptr ctx = key.MyProvider().CreateVerifierPublicCtx(
    key.GetPrimitiveId());
    ctx->Reset(params);
    ctx->SetKey(key);
    return ctx;
}
```

Listing 7.17: Digital signature verifier context creation and initialization

```
/// @brief Factory that creates and initializes a stream cipher context
    according to provided key and IV objects.
/// @param[in] key  Symmetric key object
/// @param[in] iv  Initialization Vector (IV) that should be loaded to the
    target context (object content hidden from the application)
/// @param[in] directTransform  Flag that controls direction of the
    transformation: @c true - encrypt, @c false - decrypt
/// @param[in] params  Optional pointer to a "domain parameters" object (
    cipher-specific extra configuration)
```

```
/// @return Unique smart pointer to the created context instance
/// @note Key object @b must have complete specification of the target
    algorithm!
StreamCipherCtx::Uptr initCipher(const SymmetricKey & key, const SecretSeed
     & iv, bool directTransform = true
     , DomainParameters::Sptrc params = nullptr)
{
     StreamCipherCtx::Uptr ctx = key.MyProvider().CreateStreamCipherCtx(key.
    GetPrimitiveId());
     ctx->Reset(params);
     ctx->SetKey(key, directTransform);
     ctx->Start(iv);
     return ctx;
}
```

Listing 7.18: Stream Cipher context creation and initialization

```
/// @brief Factory that creates and initializes an authenticated stream
    cipher context according to provided key and IV objects.
/// @param[in] key  Symmetric key object
/// @param[in] iv  Initialization Vector that should be loaded to the
    target context (value is default or supplied by the application)
/// @param[in] directTransform  Flag that controls direction of the
    transformation: @c true - encrypt, @c false - decrypt
/// @param[in] params  Optional pointer to a "domain parameters" object (
    cipher-specific extra configuration)
/// @return Unique smart pointer to the created context instance
/// @note Key object @b must have complete specification of the target
    algorithm!
AuthnStreamCipherCtx::Uptr initAuthCipher(const SymmetricKey & key,
    ReadOnlyMemRegion iv = ReadOnlyMemRegion()
     , bool directTransform = true, DomainParameters::Sptrc params = nullptr
    )
{
     AuthnStreamCipherCtx::Uptr ctx = key.MyProvider().
    CreateAuthnStreamCipherCtx(key.GetPrimitiveId());
     ctx->Reset(params);
     ctx->SetKey(key, directTransform);
     ctx->Start(iv);
     return ctx;
}
```

Listing 7.19: Authenticated Stream Cipher context creation and initialization

```
/// @brief Factory that creates and initializes a message authentication
    code (MAC) context according to provided key and IV objects.
/// @param[in] key  Symmetric key object
/// @param[in] iv  Initialization Vector that should be loaded to the
    target context (value is default or supplied by the application)
/// @param[in] params  Optional pointer to a "domain parameters" object (
    MAC-specific extra configuration)
/// @return Unique smart pointer to the created context instance
/// @note Key object @b must have complete specification of the target
    algorithm!
MessageAuthnCodeCtx::Uptr initMac(const SymmetricKey & key
     , ReadOnlyMemRegion iv = ReadOnlyMemRegion(), DomainParameters::Sptrc
    params = nullptr)
```

```
{
    MessageAuthnCodeCtx::Uptr ctx = key.MyProvider().
   CreateMessageAuthnCodeCtx(key.GetPrimitiveId());
    ctx->Reset(params);
    ctx->SetKey(key);
    ctx->Start(iv);
    return ctx;
}
```

Listing 7.20: Message Authentication Code context creation and initialization

7.6.4 Finalization routines

```
/// @brief Finalize digest calculation and check the result by the expected
    value.
/// @param[in] digestCtx  Initialized digest calculation context (MAC/HMAC/
   Hash)
/// @param[in] digest  Expected digest value (template for comparison)
/// @param[in] offset  Optional offset of the first byte of calculated
   digest that should be used for the comparison
/// @return @c true if the provided digest is equal to the calculated one
bool checkDigest(BufferedDigest & digestCtx, ReadOnlyMemRegion digest,
   size_t offset = 0)
{
    digestCtx.Finish();
    return digestCtx.Compare(digest, offset);
}
```

Listing 7.21: Finalization of digest calculation and check the result

```
/// @brief Finalize digest calculation and return full or truncated result.
/// @param[in] digestCtx  Initialized digest calculation context (MAC/HMAC/
   Hash)
/// @param[in] digest  Buffer for placing the output value
/// @param[in] offset  Optional offset of the first byte of calculated
   digest that should be returned
/// @return Number of bytes actually placed to the output buffer
size_t getDigest(BufferedDigest & digestCtx, WritableMemRegion digest,
   size_t offset = 0)
{
    digestCtx.Finish();
    return digestCtx.GetDigest(digest, offset);
}
```

Listing 7.22: Finalization of digest calculation and getting the result (variant 1)

```
/// @brief Finalize digest calculation and return full or truncated result
   as a vector.
/// @note Similar to the previous one, but cares about memory management (
   @see getDigest(BufferedDigest&, WritableMemRegion, size_t)).
/// @param[in] digestCtx  Initialized digest calculation context (MAC/HMAC/
   Hash)
/// @param[in] offset  Optional offset of the first byte of calculated
   digest that should be returned
/// @return A @c Byte Vector container that keeps the requested fragmnent
   of calculated digest.
```

```
ByteVectorT<> getDigest(BufferedDigest & digestCtx, size_t offset = 0)
{
    ByteVectorT<> digest(digestCtx.GetDigestSize());
    digestCtx.Finish();
    digestCtx.GetDigest(digest, offset);
    return digest;
}
```

Listing 7.23: Finalization of digest calculation and getting the result (variant 1)

7.6.5 Data processing routines

```
/// @brief Generic encryption/decryption of a "data stream" in any mode of
    operation
/// @param[in] cipherCtx  Cipher context already initialized for execution
    of encryption or decryption
void cryptDataStream(StreamCipherCtx & cipherCtx)
{
    static const size_t c_factor = 64;
    const size_t bs = cipherCtx.GetBlockSize();
    std::vector<Byte> buffer(bs * 2 * c_factor);
    WritableMemRegion src, dst, buf = ara::core::MakeSpan(buffer);
    if(cipherCtx.IsBytewiseMode())
    {
        src = buf;
        dst = buf;
    }
    else
    {
        src = buf.subspan(0, buf.size() / 2 - bs);
        dst = buf.subspan(src.size(), buf.size() - src.size());
    }
    while(size_t inChunkSize = readData(src))
    {
        ReadOnlyMemRegion inChunk = src.subspan(0, inChunkSize);
        size_t outChunkSize = cipherCtx.ProcessBytes(dst, inChunk);
        ReadOnlyMemRegion outChunk = dst.subspan(0, outChunkSize);
        writeData(outChunk);
    }
    if(!cipherCtx.IsBytewiseMode())
    {
        ReadOnlyMemRegion emptyChunk = src.subspan(0, 0);
        size_t lastChunkSize = cipherCtx.FinishBytes(dst, emptyChunk);
        ReadOnlyMemRegion lastChunk = dst.subspan(0, lastChunkSize);
        writeData(lastChunk);
    }
}
```

Listing 7.24: Generic encryption/decryption of a "data stream"

```
/// @brief Optimized encryption/decryption of a "data chunks sequence" in a
     block-wise mode of operation
/// @param[in] cipherCtx  Cipher context already initialized for execution
    of encryption or decryption
void cryptChunksBlockwise(StreamCipherCtx & cipherCtx)
{
    const size_t lastChunk = getChunksCounter().cTotal - 1;
    const size_t firstAlignedConfidentialChunk = getChunksCounter().cPublic;
    const size_t firstNonAlignedConfidentialChunk = getChunksCounter().cPublic + getChunksCounter().cAligned;
    size_t i;
    for(i = firstAlignedConfidentialChunk; i < firstNonAlignedConfidentialChunk; i++)
    {
        cipherCtx.ProcessBlocks(getOutChunkBuffer(i), getInDataChunk(i));
    }
    for(i = firstNonAlignedConfidentialChunk; i < lastChunk; i++)
    {
        size_t outSize = cipherCtx.ProcessBytes(getOutChunkBuffer(i), getInDataChunk(i));
        setOutChunkSize(i, outSize);
    }
    size_t outSize = cipherCtx.FinishBytes(getOutChunkBuffer(lastChunk), getInDataChunk(lastChunk));
    setOutChunkSize(lastChunk, outSize);
}
```

Listing 7.25: Encryption/Decryption of a "data chunks sequence" in a block-wise mode

```
/// @brief Generic encryption/decryption of a "data chunks sequence" in a byte-wise mode of operation
/// @param[in] cipherCtx  Cipher context already initialized for execution of encryption or decryption
void cryptChunksBytewise(StreamCipherCtx & cipherCtx)
{
    const size_t totalChunks = getChunksCounter().cTotal;
    const size_t firstConfidentialChunk = getChunksCounter().cPublic;
    for(size_t i = firstConfidentialChunk; i < totalChunks; i++)
    {
        (void)cipherCtx.ProcessBytes(getOutChunkBuffer(i), getInDataChunk(i));
    }
}
```

Listing 7.26: Encryption/Decryption of a "data chunks sequence" in a byte-wise mode

```
/// @brief Generic processing of public "data chunks" in a sequence for authentication.
/// @param[in] digestCtx  Initialized digest calculation context (MAC/HMAC/Hash)
/// @param[in] publicChunks  Number of leading public chunks in the sequence that should be authenticated
void processPublicChunks(BufferedDigest & digestCtx)
{
    const size_t publicChunks = getChunksCounter().cPublic;
    for(size_t i = 0; i < publicChunks; i++)
    {
        (void)digestCtx.Update(getInDataChunk(i));
    }
}
```

Listing 7.27: Processing of public "data chunks" for authentication

```
/// @brief Optimized encryption/decryption of a fixed-size "file" in a
    block-wise mode of operation.
/// @param[in] cipherCtx  Cipher context already initialized for execution
    of encryption or decryption
void cryptFileBlockwise(StreamCipherCtx & cipherCtx)
{
    static const size_t c_factor = 64;
    const size_t bs = cipherCtx.GetBlockSize();
    std::vector<Byte> buffer(bs * c_factor);
    WritableMemRegion buf = ara::core::MakeSpan(buffer);
    const size_t totalSize = getDataSize();
    const size_t n = totalSize / buffer.size();
    for(int i = 0; i < n; i++)
    {
        readData(buf);
        cipherCtx.ProcessBlocks(buf);
        writeData(buf);
    }
    size_t restSize = totalSize % buffer.size();
    if(restSize)
    {
        const size_t partialBlock = restSize % bs;
        restSize -= partialBlock;
        if(restSize)
        {
            buf = ara::core::MakeSpan(buffer.data(), restSize);
            readData(buf);
            cipherCtx.ProcessBlocks(buf);
            writeData(buf);
        }
        if(partialBlock)
        {
            WritableMemRegion lastIn = ara::core::MakeSpan(buffer.data(),
    partialBlock);
            buf = ara::core::MakeSpan(buffer.data() + bs, bs);
            readData(lastIn);
            size_t outSize = cipherCtx.FinishBytes(buf, lastIn);
            ReadOnlyMemRegion lastOut = buf.subspan(0, outSize);
            writeData(lastOut);
        }
    }
}
```

Listing 7.28: Optimized encryption/decryption of a fixed-size "file"

7.6.6 Demonstration of the whole primitive's usage sequence

```
/// @brief Demonstrate whole usage sequence of a stream cipher for
    encryption/decryption of a data stream.
/// @param[in] key  Symmetric key that should be used for encryption/
    decryption
/// @param[in] iv  Initialization Vector (IV) that should be used for
    encryption/decryption
/// @param[in] directTransform  Flag that controls direction of the
    transformation: @c true - encrypt, @c false - decrypt
```

```
/// @note @c directTransform argument is ignored in some modes of
    operations, like OFB, CTR.
void useCipher(const SymmetricKey & key, const SecretSeed & iv, bool
    directTransform = true)
{
    StreamCipherCtx::Uptr ctx = createCipher(key);  ///< Context Creation
    initCipher(*ctx, key, iv, directTransform);  ///< Context
    Initialization
    cryptDataStream(*ctx);  ///< Data processing (including Finalization)
}
```

Listing 7.29: Encryption/Decryption by Stream Cipher

```
/// @brief Demonstrate whole usage sequence of an authenticated stream
    cipher for encryption & MAC-calculation of a data stream.
/// @param[in] key  Symmetric key that should be used for encryption &
    authentication
/// @param[in] iv  Initialization Vector (IV) that should be used for
    encryption & authentication
/// @return Full MAC-value calculated by the transformation.
ByteVectorT<> useAuthCipherDirect(const SymmetricKey & key, const
    SecretSeed & iv)
{
    AuthnStreamCipherCtx::Uptr ctx = createAuthCipher(key);  ///< Context
    Creation
    initCipher(*ctx, key, iv, true);  ///< Context Initialization
    processPublicChunks(*ctx);  ///< Public (associated) data processing
    cryptChunksBlockwise(*ctx);  ///< Confidential data processing
    return getDigest(*ctx);  ///< Finalization
}
```

Listing 7.30: Encryption and MAC-calculation by Authenticated Stream Cipher

```
/// @brief Demonstrate whole usage sequence of an authenticated stream
    cipher for decryption & MAC-verification of a data stream.
/// @param[in] key  Symmetric key that should be used for decryption &
    authentication
/// @param[in] iv  Initialization Vector (IV) that should be used for
    decryption & authentication
/// @param[in] mac  Expected MAC value.
/// @return @c true if MAC was verified successfully or @c false otherwise.
bool useAuthCipherInverse(const SymmetricKey & key, const SecretSeed & iv,
    ReadOnlyMemRegion mac)
{
    AuthnStreamCipherCtx::Uptr ctx = createAuthCipher(key);  ///< Context
    Creation
    initCipher(*ctx, key, iv, false);  ///< Context Initialization
    processPublicChunks(*ctx);  ///< Public (associated) data processing
    cryptChunksBytewise(*ctx);  ///< Confidential data processing
    return checkDigest(*ctx, mac);  ///< Finalization
}
```

Listing 7.31: Decryption and MAC-verification by Authenticated Stream Cipher

```
/// @brief Demonstrate whole usage sequence of a hash function calculation
    for a data chunks sequence.
/// @param[in] algName  Name of the target hash-function algorithm
/// @return Full hash-value calculated by the transformation
ByteVectorT<> calcHash(StringView algName)
{
    HashFunctionCtx::Uptr ctx = createHashFunc(algName);  ///< Context
   Creation
    initHashFunc(*ctx);  ///< Context Initialization
    processPublicChunks(*ctx);  ///< Data processing (hashing)
    return getDigest(*ctx);  ///< Finalization
}
```

Listing 7.32: Hash Function calculation

```
/// @brief Demonstrate whole usage sequence of a hash function verification
    of a data chunks sequence.
/// @param[in] algName  Name of the target hash-function algorithm
/// @param[in] hash  Expected hash value
/// @return @c true if the hash value was verified successfully or @c false
    otherwise
bool verifyHash(StringView algName, ReadOnlyMemRegion hash)
{
    HashFunctionCtx::Uptr ctx = createHashFunc(algName);  ///< Context
   Creation
    initHashFunc(*ctx);  ///< Context Initialization
    processPublicChunks(*ctx);  ///< Data processing (hashing)
    return checkDigest(*ctx, hash);  ///< Finalization
}
```

Listing 7.33: Hash Function verification

```
/// @brief Demonstrate whole usage sequence of a MAC calculation for a data
    chunks sequence.
/// @param[in] key  Symmetric key that should be used for authentication
/// @param[in] iv  Optional Initialization Vector (IV) that should be used
   for authentication
/// @return Full MAC-value calculated by the transformation
ByteVectorT<> calcMAC(const SymmetricKey & key, ReadOnlyMemRegion iv =
   ReadOnlyMemRegion())
{
    MessageAuthnCodeCtx::Uptr ctx = initMac(key, iv);  ///< Context
   Creation & Initialization
    processPublicChunks(*ctx);  ///< Data processing (MAC calculation)
    return getDigest(*ctx);  ///< MAC Finalization
}
```

Listing 7.34: Message Authentication Code calculation

```
/// @brief Demonstrate whole usage sequence of a MAC verification of a data
    chunks sequence.
/// @param[in] mac  Expected MAC value.
/// @param[in] key  Symmetric key that should be used for authentication
/// @param[in] iv  Optional Initialization Vector (IV) that should be used
   for authentication
/// @return @c true if the MAC was verified successfully or @c false
   otherwise
bool verifyMAC(ReadOnlyMemRegion mac, const SymmetricKey & key,
   ReadOnlyMemRegion iv = ReadOnlyMemRegion())
{
    MessageAuthnCodeCtx::Uptr ctx = initMac(key, iv);  ///< Context Creation & Initialization
    processPublicChunks(*ctx);  ///< Data processing (MAC calculation)
    return checkDigest(*ctx, mac);  ///< MAC Finalization
}
```

Listing 7.35: Message Authentication Code verification

```
/// @brief Demonstrate whole usage sequence of a digital signature calculation of a data chunks sequence.
/// @param[in] key  Private (signature) key object
/// @param[in] params  Optional pointer to a signature domain parameters object
/// @return Produced @c Signature object.
Signature::Uptrc signData(const PrivateKey & key, DomainParameters::Sptrc params = nullptr)
{
    SignerPrivateCtx::Uptr signerCtx = initSigner(key, params);  ///< Signer Context Creation & Initialization

    CryptoProvider & cp = signerCtx->MyProvider();
    CryptoAlgId hashAlgId = signerCtx->GetRequiredHashAlgId();
    HashFunctionCtx::Uptr hashCtx = cp.CreateHashFunctionCtx(hashAlgId);  ///< Hash Context Creation

    hashCtx->Start();   ///< Hash Context Initialization
    processPublicChunks(*hashCtx);  ///< Data processing (hashing)
    hashCtx->Finish();  ///< Hash Calculation Finalization

    return signerCtx->Sign(*hashCtx);  ///< Signature calculation (Finalization of the whole process)
}
```

Listing 7.36: Digital Signature calculation

```
/// @brief Demonstrate whole usage sequence of a digital signature calculation of a data chunks sequence.
/// @param[in] sig  Signature object
/// @param[in] key  Public (verification) key object
/// @param[in] params  Optional pointer to a signature domain parameters object
/// @return @c true if the signature was verified successfully or @c false otherwise
bool verifySignedData(const Signature & sig, const PublicKey & key, DomainParameters::Sptrc params = nullptr)
{
    VerifierPublicCtx::Uptr verifierCtx = initVerifier(key, params);  ///< Virifier Context Creation & Initialization

    /// Hash Context Creation:
    HashFunctionCtx::Uptr hashCtx = verifierCtx->MyProvider().CreateHashFunctionCtx(verifierCtx->GetRequiredHashAlgId());

    hashCtx->Start();   ///< Hash Context Initialization
    processPublicChunks(*hashCtx);  ///< Data processing (hashing)
    hashCtx->Finish();  ///< Hash Calculation Finalization

    return verifierCtx->Verify(*hashCtx, sig);  ///< Signature verification
     (Finalization of the whole process)
}
```

Listing 7.37: Digital Signature verification

8 Crypto API Reference

8.1 Modules

Here is a list of all modules:

8.2 ara::crypto Namespace Reference

Namespaces

- cryp
- keys
- x509

Classes

- class AccessViolationException
- class BadAllocException
- class BadObjectTypeException
- struct CryptoObjectUid
- struct CustomDeleter

- class CustomDisposable
- class InvalidArgumentException
- class InvalidUsageOrderException
- class LogicException
- class ProviderInfo
- class ResourceException
- class RuntimeException
- class SecurityErrorDomain
- class SecurityException
- class Serializable
- class TrustedContainer
- class UnexpectedValueException
- class UnsupportedException
- class UsageViolationException
- struct Uuid

Typedefs

- using ActorUid = Uuid
- using CryptoProviderUid = Uuid
- using LogicalSlotUid = Uuid
- using CryptoAlgId = std::uint64_t
- using AllowedUsageFlags = std::uint32_t
- using Guid = Uuid
- using Byte = std::uint8_t
- using ReadWriteMemRegion = ara::core::Span< Byte >
- using WritableMemRegion = ara::core::Span< Byte >
- using ReadOnlyMemRegion = ara::core::Span< const Byte >
- using DefBytesAllocator = std::allocator< std::uint8_t >
- template<class Alloc = DefBytesAllocator>
  using ByteVectorT = ara::core::Vector< std::uint8_t, Alloc >

Enumerations

Functions

- constexpr bool operator== (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool operator< (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool operator> (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool operator!= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool operator<= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool operator>= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr ara::core::ErrorDomain const & GetSecurityErrorDomain ()
- constexpr ara::core::ErrorCode MakeErrorCode (SecurityErrorDomain::Errc code, ara::core::ErrorDomain::SupportDataType data, char const ∗message=nullptr)
- constexpr bool operator== (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool operator< (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool operator> (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool operator!= (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool operator<= (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool operator>= (const Uuid &lhs, const Uuid &rhs) noexcept
- bool operator== (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool operator< (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool operator> (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool operator!= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool operator<= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool operator>= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept

Variables

- const CryptoProviderUid kAnyCryptoProvider = Uuid()
- const CryptoAlgId kAlgIdUndefined = 0ULL
- const CryptoAlgId kAlgIdAny = kAlgIdUndefined
- const CryptoAlgId kAlgIdDefault = kAlgIdUndefined
- const CryptoAlgId kAlgIdNone = kAlgIdUndefined
- const AllowedUsageFlags kAllowPrototypedOnly = 0L
- const AllowedUsageFlags kAllowDataEncryption = 0x0001L
- const AllowedUsageFlags kAllowDataDecryption = 0x0002L
- const AllowedUsageFlags kAllowSignature = 0x0004L
- const AllowedUsageFlags kAllowVerification = 0x0008L
- const AllowedUsageFlags kAllowKeyAgreement = 0x0010L
- const AllowedUsageFlags kAllowKeyDiversify = 0x0020L
- const AllowedUsageFlags kAllowDrngInit = 0x0040L
- const AllowedUsageFlags kAllowKdfMaterial = 0x0080L
- const AllowedUsageFlags kAllowKeyExporting = 0x0100L
- const AllowedUsageFlags kAllowKeyImporting = 0x0200L
- const AllowedUsageFlags kAllowExactModeOnly = 0x8000L
- const AllowedUsageFlags kAllowDerivedDataEncryption = kAllowDataEncryption << 16
- const AllowedUsageFlags kAllowDerivedDataDecryption = kAllowDataDecryption << 16
- const AllowedUsageFlags kAllowDerivedSignature = kAllowSignature << 16
- const AllowedUsageFlags kAllowDerivedVerification = kAllowVerification << 16
- const AllowedUsageFlags kAllowDerivedKeyDiversify = kAllowKeyDiversify << 16
- const AllowedUsageFlags kAllowDerivedDrngInit = kAllowDrngInit << 16
- const AllowedUsageFlags kAllowDerivedKdfMaterial = kAllowKdfMaterial << 16
- const AllowedUsageFlags kAllowDerivedKeyExporting = kAllowKeyExporting << 16
- const AllowedUsageFlags kAllowDerivedKeyImporting = kAllowKeyImporting << 16
- const AllowedUsageFlags kAllowDerivedExactModeOnly = kAllowExactModeOnly << 16
- const AllowedUsageFlags kAllowKdfMaterialAnyUsage

Detailed Description

Namespace of AUTOSAR Adaptive Platform Crypto API. The Crypto API defines public interfaces provided by the Crypto Stack. Crypto API includes 3 functional sub-domains presented by correspondent provider types:

- Crypto Provider (multiple instances in the Stack), namespace `ara::crypto::cryp`
- Key Storage Provider (single instance in the Stack), namespace `ara::crypto::keys`
- X.509 Provider (single instance in the Stack), namespace `ara::crypto::x509`.

But common part of the Crypto API is defined directly in this "root" namespace `ara::crypto`.

8.2.1 ara::crypto::cryp Namespace Reference

Classes

- class AuthnStreamCipherCtx
- class BlockCryptor
- class BufferedDigest
- class CryptoContext
- class CryptoObject
- class CryptoPrimitiveId
- class CryptoProvider
- class DecryptorPrivateCtx
- class DomainParameters
- class EncryptorPublicCtx
- class HashFunctionCtx
- class Key
- class KeyAgreementPrivateCtx
- class KeyDecapsulatorPrivateCtx
- class KeyDerivationFunctionCtx
- class KeyDiversifierCtx
- class KeyedContext
- class KeyEncapsulator
- class KeyEncapsulatorPublicCtx

- class KeyMaterial
- class MessageAuthnCodeCtx
- class MsgRecoveryPublicCtx
- class PasswordCache
- class PasswordHash
- class PrivateKey
- class PrivateKeyContext
- class PublicKey
- class PublicKeyContext
- class RandomGeneratorCtx
- class RestrictedUseObject
- class SecretSeed
- class SigEncodePrivateCtx
- class Signature
- class SignatureHandler
- class SignerPrivateCtx
- class StreamCipherCtx
- class StreamStarter
- class SymmetricBlockCipherCtx
- class SymmetricKey
- class SymmetricKeyContext
- class SymmetricKeyWrapperCtx
- class VerifierPublicCtx
- class X509AlgorithmId
- class X509CertRequest
- class X509PublicKeyInfo
- class X509RequestSignerCtx
- class X509Signature

Typedefs

- using ReservedContextIndex = std::size_t
- using ReservedObjectIndex = std::size_t

Enumerations

Functions

- ara::core::Result< CryptoProvider::Sptr > LoadCryptoProvider (const CryptoProviderUid *providerUid=nullptr) noexcept

Variables

- static const ReservedContextIndex kAllocContextOnHeap = static_cast<ReservedContextIndex>(-1)
- static const ReservedObjectIndex kAllocObjectOnHeap = static_cast<ReservedObjectIndex>(-1)

Detailed Description

Namespace of Crypto Provider API. Crypto Provider API exposes complete set of interfaces for implementation of all cryptographic primitives supported by the Crypto Stack. Single Crypto Stack may support multiple Crypto Providers simultaneously. Each Crypto Provider represents single isolated implementation (software-based or HSM-based). All public interfaces of a Crypto Provider are defined in this namespace.

8.2.2 ara::crypto::keys Namespace Reference

Classes

- struct KeySlotContentProps
- struct KeySlotPrototypeProps
- class KeyStorageProvider
- class UpdatesObserver
- struct UserPermissions

Typedefs

- using SlotNumber = std::size_t
- using TransactionId = std::uint64_t
- using TransactionScope = ara::core::Vector< SlotNumber >

Enumerations

Functions

- ara::core::Result< KeyStorageProvider::Sptr > LoadKeyStorageProvider () noexcept
- constexpr bool operator== (const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) noexcept
- constexpr bool operator!= (const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) noexcept
- constexpr bool operator== (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept
- constexpr bool operator!= (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept
- constexpr bool operator== (const UserPermissions &lhs, const UserPermissions &rhs) noexcept
- constexpr bool operator!= (const UserPermissions &lhs, const UserPermissions &rhs) noexcept

Variables

- const SlotNumber kInvalidSlot = static_cast<SlotNumber>(-1LL)

Detailed Description

Namespace of Key Storage Provider API. Key Storage Provider API exposes complete set of interfaces for saving, loading and management of key material and related cryptographic objects in a protected persistent storage. All public interfaces of the Key Storage Provider are defined in this namespace.

8.2.3 ara::crypto::x509 Namespace Reference

Classes

- class BasicCertInfo
- class Certificate
- class CertSignRequest
- class OcspRequest
- class OcspResponse

- class X509DN
- class X509Extensions
- class X509Provider

Enumerations

Functions

- ara::core::Result< X509Provider::Sptr > LoadX509Provider () noexcept

Detailed Description

Namespace of X.509 Provider API. X.509 Provider API exposes complete set of interfaces for the client-side support of X.509 compliant Public Key Infrastructure (PKI). All public interfaces of the X.509 Provider are defined in this namespace.

8.3 ara/crypto Directory Reference

Directories

- Directory ara/crypto/common
- Directory ara/crypto/cryp
- Directory ara/crypto/keys
- Directory ara/crypto/x509

Detailed Description

This directory (together with it's sub-directories) contains complete specification of the Crypto API.

8.3.1 ara/crypto/common Directory Reference

Files

- file **base_id_types.h**
- file **crypto_object_uid.h**
- file **custom_disposable.h**

- file **exceptions.h**
- file **guid.h**
- file **mem_region.h**
- file **provider_info.h**
- file **serializable.h**
- file **std_api.h**
- file **trusted_container.h**

Detailed Description

This directory contains definitions of common types shared by all other sub-domains of Crypto API. These definitions are described in the section Common API.

8.3.2 ara/crypto/cryp Directory Reference

Files

- file **authn_stream_cipher_ctx.h**
- file **block_cryptor.h**
- file **buffered_digest.h**
- file **crypto_context.h**
- file **crypto_object.h**
- file **crypto_primitive_id.h**
- file **crypto_provider.h**
- file **decryptor_private_ctx.h**
- file **domain_parameters.h**
- file **encryptor_public_ctx.h**
- file **entry_point.h**
- file **hash_function_ctx.h**
- file **key.h**
- file **key_agreement_private_ctx.h**
- file **key_decapsulator_private_ctx.h**
- file **key_derivation_function_ctx.h**
- file **key_diversifier_ctx.h**
- file **key_encapsulator.h**
- file **key_encapsulator_public_ctx.h**

- file **key_material.h**
- file **key_type.h**
- file **keyed_context.h**
- file **memory_pool.h**
- file **message_authn_code_ctx.h**
- file **msg_recovery_public_ctx.h**
- file **password_cache.h**
- file **password_hash.h**
- file **private_key.h**
- file **private_key_context.h**
- file **public_key.h**
- file **public_key_context.h**
- file **random_generator_ctx.h**
- file **restricted_use_object.h**
- file **secret_seed.h**
- file **sig_encode_private_ctx.h**
- file **signature.h**
- file **signature_handler.h**
- file **signer_private_ctx.h**
- file **stream_cipher_ctx.h**
- file **stream_starter.h**
- file **symmetric_block_cipher_ctx.h**
- file **symmetric_key.h**
- file **symmetric_key_context.h**
- file **symmetric_key_wrapper_ctx.h**
- file **verifier_public_ctx.h**
- file **x509_algorithm_id.h**
- file **x509_cert_request.h**
- file **x509_public_key_info.h**
- file **x509_request_signer_ctx.h**
- file **x509_signature.h**

Detailed Description

This directory contains complete specification of the Crypto Provider API.

Note

> In order to use the whole Crypto Provider API it is enough to include only one header file: `"ara/crypto/cryp/entry_point.h"`.

8.3.3 ara/crypto/keys Directory Reference

Files

- file **elementary_types.h**
- file **entry_point.h**
- file **key_slot_content_props.h**
- file **key_slot_prototype_props.h**
- file **key_storage_provider.h**
- file **updates_observer.h**
- file **user_permissions.h**

Detailed Description

This directory contains complete specification of the Key Storage Provider API.

Note

> In order to use the whole Key Storage Provider API it is enough to include only one header file: `"ara/crypto/keys/entry_point.h"`.

8.3.4 ara/crypto/x509 Directory Reference

Files

- file **basic_cert_info.h**
- file **cert_sign_request.h**
- file **certificate.h**
- file **entry_point.h**
- file **ocsp_request.h**
- file **ocsp_response.h**
- file **x509_dn.h**
- file **x509_extensions.h**
- file **x509_provider.h**

Detailed Description

This directory contains complete specification of the X.509 Provider API.

Note

> In order to use the whole X.509 Provider API it is enough to include only one header file: "`ara/crypto/x509/entry_point.h`".

8.4 Crypto API

Modules

- Common API
- Crypto Provider API
- Key Storage Provider API
- X.509 Provider API

Directories

- Directory ara/crypto
- Directory ara/crypto/common
- Directory ara/crypto/cryp
- Directory ara/crypto/keys
- Directory ara/crypto/x509

Namespaces

- ara::crypto

Detailed Description

Adaptive Platform Crypto API represents public API provided by CryptoStack for Adaptive Applications. This API incorporates a set of 3 functional domains:

- Cryptographic primitives implementation
- Secure storage of cryptographic key material
- Certificates management

8.5 Common API

Modules

- Exceptions & Error Codes
- Simple common types
- Basic common interfaces

Detailed Description

Common part of Crypto API shared by all functional sub-domains of CryptoStack.

8.6 Simple common types

Modules

- Reserved Crypto Algorithm ID
- Allowed Usage bit-flags

Classes

- struct ara::crypto::CryptoObjectUid
- struct ara::crypto::Uuid

Typedefs

- using ara::crypto::ActorUid = Uuid
- using ara::crypto::CryptoProviderUid = Uuid
- using ara::crypto::LogicalSlotUid = Uuid
- using ara::crypto::CryptoAlgId = std::uint64_t
- using ara::crypto::AllowedUsageFlags = std::uint32_t
- using ara::crypto::Guid = Uuid
- using ara::crypto::Byte = std::uint8_t
- using ara::crypto::ReadWriteMemRegion = ara::core::Span< Byte >
- using ara::crypto::WritableMemRegion = ara::core::Span< Byte >
- using ara::crypto::ReadOnlyMemRegion = ara::core::Span< const Byte >
- using ara::crypto::DefBytesAllocator = std::allocator< std::uint8_t >
- template<class Alloc = DefBytesAllocator>
  using ara::crypto::ByteVectorT = ara::core::Vector< std::uint8_t, Alloc >

Enumerations

Functions

- constexpr bool ara::crypto::operator== (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool ara::crypto::operator< (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept

- constexpr bool ara::crypto::operator> (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool ara::crypto::operator!= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool ara::crypto::operator<= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool ara::crypto::operator>= (const CryptoObjectUid &lhs, const CryptoObjectUid &rhs) noexcept
- constexpr bool ara::crypto::operator== (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool ara::crypto::operator< (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool ara::crypto::operator> (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool ara::crypto::operator!= (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool ara::crypto::operator<= (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool ara::crypto::operator>= (const Uuid &lhs, const Uuid &rhs) noexcept
- constexpr bool ara::crypto::Uuid::IsNil () const noexcept

Variables

- const CryptoProviderUid ara::crypto::kAnyCryptoProvider = Uuid()

Detailed Description

Group of simple types completely defined in header files and shared by all functional sub-domains of CryptoStack.

8.6.1 Class Documentation

8.6.1.1 struct ara::crypto::CryptoObjectUid

[SWS_CRYPT_10100]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::crypto::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, RS_CRYPTO_02404)*

Public Member Functions

- constexpr bool HasSameSourceAs (const CryptoObjectUid &anotherId) const noexcept
- constexpr bool HasEarlierVersionThan (const CryptoObjectUid &anotherId) const noexcept
- constexpr bool HasLaterVersionThan (const CryptoObjectUid &anotherId) const noexcept
- constexpr bool IsNil () const noexcept

Public Attributes

- Guid mGeneratorUid
- std::uint64_t mVersionStamp = 0ULL

8.6.1.1.1 Member Function Documentation

8.6.1.1.1.1 constexpr bool ara::crypto::CryptoObjectUid::HasSameSourceAs (const CryptoObjectUid & *anotherId*) const `[noexcept]`

[SWS_CRYPT_10111]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::CryptoObjectUid::HasSameSourceAs(const CryptoObjectUid &anotherId)	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`inline 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)*

8.6.1.1.1.2 constexpr bool ara::crypto::CryptoObjectUid::HasEarlierVersionThan (const CryptoObjectUid & *anotherId*) const `[noexcept]`

[SWS_CRYPT_10112]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::CryptoObjectUid::HasEarlierVersionThan(const CryptoObjectUid &anotherId)	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`inline 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)*

8.6.1.1.1.3 constexpr bool ara::crypto::CryptoObjectUid::HasLaterVersionThan (const CryptoObjectUid & *anotherId*) const `[noexcept]`

[SWS_CRYPT_10113]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::CryptoObjectUid::HasLaterVersionThan(const CryptoObjectUid &anotherId)	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`inline 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)*

8.6.1.1.1.4 constexpr bool ara::crypto::CryptoObjectUid::IsNil ( ) const `[noexcept]`

[SWS_CRYPT_10114]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::CryptoObjectUid::IsNil()	
Scope:	struct ara::crypto::CryptoObjectUid	
Syntax:	`inline constexpr 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)*

8.6.1.1.2 Member Data Documentation

8.6.1.1.2.1 Guid ara::crypto::CryptoObjectUid::mGeneratorUid

[SWS_CRYPT_10101]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::CryptoObjectUid::mGeneratorUid
Scope:	struct ara::crypto::CryptoObjectUid
Type:	Guid
Syntax:	`Guid ara::crypto::CryptoObjectUid::mGeneratorUid;`
Header file:	#include "ara/crypto/common/crypto_object_uid.h"
Description:	UUID of a generator that has produced this COUID. This UUID can be associated with HSM, physical host/ECU or VM.

⌋*(RS_CRYPTO_02006)*

8.6.1.1.2.2 std::uint64_t ara::crypto::CryptoObjectUid::mVersionStamp = 0ULL

[SWS_CRYPT_10102]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::CryptoObjectUid::mVersionStamp
Scope:	struct ara::crypto::CryptoObjectUid
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::crypto::CryptoObjectUid::mVersionStamp= 0ULL;`
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)*

8.6.1.2 struct ara::crypto::Uuid

[SWS_CRYPT_10400]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::crypto::Uuid
Scope:	namespace ara::crypto
Syntax:	`struct Uuid {...};`
Header file:	#include "ara/crypto/common/guid.h"
Description:	Definition of Universally Unique Identifier (UUID) or Globally Unique Identifier (GUID) type.
Notes:	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_02404)*

Public Member Functions

- constexpr bool IsNil () const noexcept

8.6.2 Typedef Documentation

8.6.2.1 using ara::crypto::ActorUid = typedef Uuid

[SWS_CRYPT_10010]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::ActorUid
Scope:	namespace ara::crypto
Derived from:	typedef Uuid
Syntax:	`using ara::crypto::ActorUid = Uuid;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Unique ID of the "Actor" modeled process (it is a persistent UID defined on the application design phase).
Notes:	Actor UID can be associated with "User" or "Owner" process permissions.

⌋*(RS_CRYPTO_02110, RS_CRYPTO_02114, RS_CRYPTO_02404)*

8.6.2.2 using ara::crypto::CryptoProviderUid = typedef Uuid

[SWS_CRYPT_10011]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::CryptoProviderUid
Scope:	namespace ara::crypto
Derived from:	typedef Uuid
Syntax:	`using ara::crypto::CryptoProviderUid = Uuid;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Unique ID of a Crypto Provider (it is a persistent UID defined on the provider design phase).

⌋*(RS_CRYPTO_02401, RS_CRYPTO_02404)*

8.6.2.3 using ara::crypto::LogicalSlotUid = typedef Uuid

[SWS_CRYPT_10013]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::LogicalSlotUid
Scope:	namespace ara::crypto
Derived from:	typedef Uuid
Syntax:	`using ara::crypto::LogicalSlotUid = Uuid;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Logical Slot UID (it is a persistent UID defined on the application design phase).

⌋*(RS_CRYPTO_02404, RS_CRYPTO_02405)*

8.6.2.4 using ara::crypto::CryptoAlgId = typedef std::uint64_t

[SWS_CRYPT_10014]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::CryptoAlgId
Scope:	namespace ara::crypto
Derived from:	typedef std::uint64_t
Syntax:	`using ara::crypto::CryptoAlgId = std::uint64_t;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Container type of the Crypto Algorithm Identifier.

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02404)*

8.6.2.5 using ara::crypto::AllowedUsageFlags = typedef std::uint32_t

[SWS_CRYPT_10015]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::AllowedUsageFlags
Scope:	namespace ara::crypto
Derived from:	typedef std::uint32_t
Syntax:	`using ara::crypto::AllowedUsageFlags = std::uint32_t;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A container type and constant bit-flags of allowed usages of a key or a secret seed object.
Notes:	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 kAllow KeyAgreement or kAllowKeyDiversify or kAllowKeyDerivation!

⌋*(RS_CRYPTO_02111, RS_CRYPTO_02404)*

8.6.2.6 using ara::crypto::Guid = typedef Uuid

[SWS_CRYPT_10450]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::Guid
Scope:	namespace ara::crypto
Derived from:	typedef Uuid

▽

△

Syntax:	`using ara::crypto::Guid = Uuid;`
Header file:	#include "ara/crypto/common/guid.h"
Description:	The Globally Unique Identifier (GUID) is an alias of Universally Unique Identifier (UUID).

⌋*(RS_CRYPTO_02404)*

8.6.2.7 using ara::crypto::Byte = typedef std::uint8_t

[SWS_CRYPT_10030]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::Byte
Scope:	namespace ara::crypto
Derived from:	typedef std::uint8_t
Syntax:	`using ara::crypto::Byte = std::uint8_t;`
Header file:	#include "ara/crypto/common/mem_region.h"
Description:	The unified type definition for a single byte.

⌋*(RS_CRYPTO_02311)*

8.6.2.8 using ara::crypto::ReadWriteMemRegion = typedef ara::core::Span<Byte>

[SWS_CRYPT_10031]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::ReadWriteMemRegion
Scope:	namespace ara::crypto
Derived from:	typedef ara::core::Span<Byte>
Syntax:	`using ara::crypto::ReadWriteMemRegion = ara::core::Span<Byte>;`
Header file:	#include "ara/crypto/common/mem_region.h"
Description:	Read-Write Memory Region (intended for [in/out] arguments)

⌋*(RS_CRYPTO_02311)*

8.6.2.9 using ara::crypto::WritableMemRegion = typedef ara::core::Span<Byte>

[SWS_CRYPT_10032]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::WritableMemRegion
Scope:	namespace ara::crypto
Derived from:	typedef ara::core::Span<Byte>
Syntax:	`using ara::crypto::WritableMemRegion = ara::core::Span<Byte>;`
Header file:	#include "ara/crypto/common/mem_region.h"
Description:	Writable Memory Region (intended for [out] arguments)

⌋*(RS_CRYPTO_02311)*

8.6.2.10 using ara::crypto::ReadOnlyMemRegion = typedef ara::core::Span<const Byte>

[SWS_CRYPT_10033]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::ReadOnlyMemRegion
Scope:	namespace ara::crypto
Derived from:	typedef ara::core::Span<const Byte>
Syntax:	`using ara::crypto::ReadOnlyMemRegion = ara::core::Span<const Byte>;`
Header file:	#include "ara/crypto/common/mem_region.h"
Description:	Read-Only Memory Region (intended for [in] arguments)

⌋*(RS_CRYPTO_02311)*

8.6.2.11 using ara::crypto::DefBytesAllocator = typedef std::allocator<std::uint8_t>

[SWS_CRYPT_10001]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::DefBytesAllocator
Scope:	namespace ara::crypto
Derived from:	typedef std::allocator<std::uint8_t>
Syntax:	`using ara::crypto::DefBytesAllocator = std::allocator<std::uint8_t>;`
Header file:	#include "ara/crypto/common/std_api.h"
Description:	Alias of the default bytes sequences allocator

⌋*(RS_CRYPTO_02404)*

8.6.2.12 template<class Alloc = DefBytesAllocator> using ara::crypto::ByteVectorT = typedef ara::core::Vector<std::uint8_t, Alloc>

[SWS_CRYPT_10002]{DRAFT} ⌈

Kind:	type alias	
Symbol:	ara::crypto::ByteVectorT	
Scope:	namespace ara::crypto	
Derived from:	typedef ara::core::Vector<std::uint8_t, Alloc>	
Syntax:	`using ara::crypto::ByteVectorT = ara::core::Vector<std::uint8_t, Alloc>;`	
Template param:	Alloc	custom allocator of bytes sequences
Header file:	#include "ara/crypto/common/std_api.h"	
Description:	Alias of a bytes' vector template with customizable allocator	

⌋*(RS_CRYPTO_02404)*

8.6.3 Enumeration Type Documentation

8.6.3.1 enum ara::crypto::CryptoObjectType : std::uint8_t `[strong]`

[SWS_CRYPT_10016]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::CryptoObjectType	
Scope:	namespace ara::crypto	
Values:	kNone= 0	Used for empty containers (key slots) and in a case of the dependency absence.
	kUnknown= 0	Object type unknown (meaning is identical to the previous one)
	kDomainParameters= 1	cryp::DomainParameters object
	kSymmetricKey= 2	cryp::SymmetricKey object
	kPrivateKey= 3	cryp::PrivateKey object
	kPublicKey= 4	cryp::PublicKey object
	kSignature= 5	cryp::Signature object (asymmetric digital signature or symmetric MAC/HMAC)
	kPasswordHash= 6	cryp::PasswordHash object (it is a password hash diversified by a random seed)
	kSecretSeed= 7	cryp::SecretSeed object. Note: the seed cannot have an associated crypto algorithm!
	kCertSignRequest= 8	x509::CertSignRequest object. Note: X.509 is not fully supported yet!
	kCertificate= 9	x509::Certificate object. Note: X.509 is not fully supported yet!
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.
Notes:	Storage type: 8 bit unsigned integer.

⌋(*RS_CRYPTO_02004*)

8.6.3.2 enum ara::crypto::ProviderType : std::uint32_t `[strong]`

[SWS_CRYPT_10017]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::ProviderType	
Scope:	namespace ara::crypto	
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.	
Notes:	Storage type: 32 bit unsigned integer.	

⌋(*RS_CRYPTO_02401*, *RS_CRYPTO_02109*)

8.6.4 Function Documentation

8.6.4.1 constexpr bool ara::crypto::operator== (const CryptoObjectUid & *lhs*, const CryptoObjectUid & *rhs*) `[noexcept]`

[SWS_CRYPT_10150]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator==(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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_02311, RS_CRYPTO_02404)*

8.6.4.2 constexpr bool ara::crypto::operator< (const CryptoObjectUid & *lhs*, const CryptoObjectUid & *rhs*) `[noexcept]`

[SWS_CRYPT_10151]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator<(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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_02311, RS_CRYPTO_02404)*

8.6.4.3 constexpr bool ara::crypto::operator> (const CryptoObjectUid & *lhs*, const CryptoObjectUid & *rhs*) `[noexcept]`

[SWS_CRYPT_10152]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator>(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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_02311, RS_CRYPTO_02404)*

8.6.4.4 constexpr bool ara::crypto::operator!= (const CryptoObjectUid & *lhs*, const CryptoObjectUid & *rhs*) `[noexcept]`

[SWS_CRYPT_10153]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator!=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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_02311, RS_CRYPTO_02404)*

8.6.4.5 constexpr bool ara::crypto::operator<= (const CryptoObjectUid & *lhs*, const CryptoObjectUid & *rhs*) `[noexcept]`

[SWS_CRYPT_10154]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator<=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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_02311*, *RS_CRYPTO_02404*)

8.6.4.6 constexpr bool ara::crypto::operator>= (const CryptoObjectUid & *lhs*, const CryptoObjectUid & *rhs*) `[noexcept]`

[SWS_CRYPT_10155]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator>=(const CryptoObjectUid &lhs, const CryptoObjectUid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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_02311*, *RS_CRYPTO_02404*)

8.6.4.7 constexpr bool ara::crypto::operator== (const Uuid & *lhs*, const Uuid & *rhs*) `[noexcept]`

[SWS_CRYPT_10451]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::operator==(const Uuid &lhs, const Uuid &rhs)
Scope:	namespace ara::crypto
Syntax:	`inline 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/guid.h"	
Description:	Comparison operator "equal" for Uuid operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.6.4.8 constexpr bool ara::crypto::operator< (const Uuid & *lhs*, const Uuid & *rhs*) `[noexcept]`

[SWS_CRYPT_10452]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator<(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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/guid.h"	
Description:	Comparison operator "less than" for Uuid operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.6.4.9 constexpr bool ara::crypto::operator> (const Uuid & *lhs*, const Uuid & *rhs*) `[noexcept]`

[SWS_CRYPT_10453]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::operator>(const Uuid &lhs, const Uuid &rhs)
Scope:	namespace ara::crypto

▽

△

Syntax:	`inline 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/guid.h"	
Description:	Comparison operator "greater than" for Uuid operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.6.4.10 constexpr bool ara::crypto::operator!= (const Uuid & *lhs,* const Uuid & *rhs*) `[noexcept]`

[SWS_CRYPT_10454]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator!=(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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/guid.h"	
Description:	Comparison operator "not equal" for Uuid operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.6.4.11 constexpr bool ara::crypto::operator<= (const Uuid & *lhs,* const Uuid & *rhs*) `[noexcept]`

[SWS_CRYPT_10455]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator<=(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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/guid.h"	
Description:	Comparison operator "less than or equal" for Uuid operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.6.4.12 constexpr bool ara::crypto::operator>= (const Uuid & *lhs*, const Uuid & *rhs*) `[noexcept]`

[SWS_CRYPT_10456]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator>=(const Uuid &lhs, const Uuid &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline 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/guid.h"	
Description:	Comparison operator "greater than or equal" for Uuid operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.6.4.13 constexpr bool ara::crypto::Uuid::IsNil ( ) const `[noexcept]`

[SWS_CRYPT_10411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::Uuid::IsNil()	
Scope:	struct ara::crypto::Uuid	
Syntax:	`inline constexpr 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/guid.h"	
Description:	Check whether this identifier is the "Nil UUID" (according to RFC4122).	

⌋*(RS_CRYPTO_02404)*

8.6.5 Variable Documentation

8.6.5.1 const CryptoProviderUid ara::crypto::kAnyCryptoProvider = Uuid()

[SWS_CRYPT_10012]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::kAnyCryptoProvider
Scope:	namespace ara::crypto
Type:	const CryptoProviderUid
Syntax:	`const CryptoProviderUid ara::crypto::kAnyCryptoProvider= Uuid();`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Nil UID is reserved for specification of "any" (or default) Crypto Provider.

⌋*(RS_CRYPTO_02401, RS_CRYPTO_02404)*

8.7 Reserved Crypto Algorithm ID

Variables

- const CryptoAlgId ara::crypto::kAlgIdUndefined = 0ULL
- const CryptoAlgId ara::crypto::kAlgIdAny = kAlgIdUndefined
- const CryptoAlgId ara::crypto::kAlgIdDefault = kAlgIdUndefined
- const CryptoAlgId ara::crypto::kAlgIdNone = kAlgIdUndefined

Detailed Description

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.

8.7.1 Variable Documentation

8.7.1.1 const CryptoAlgId ara::crypto::kAlgIdUndefined = 0ULL

[SWS_CRYPT_13000]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::kAlgIdUndefined
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId ara::crypto::kAlgIdUndefined= 0ULL;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Algorithm ID is undefined.
Notes:	Also this value may be used in meanings: Any or Default algorithm, None of algorithms.

⌋*(RS_CRYPTO_02107)*

8.7.1.2 const CryptoAlgId ara::crypto::kAlgIdAny = kAlgIdUndefined

[SWS_CRYPT_13001]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::kAlgIdAny
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId ara::crypto::kAlgIdAny= kAlgIdUndefined;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Any Algorithm ID is allowed.

⌋*(RS_CRYPTO_02107)*

8.7.1.3 const CryptoAlgId ara::crypto::kAlgIdDefault = kAlgIdUndefined

[SWS_CRYPT_13002]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::kAlgIdDefault
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId ara::crypto::kAlgIdDefault= kAlgIdUndefined;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Default Algorithm ID (in current context/primitive).

⌋*(RS_CRYPTO_02107)*

8.7.1.4 const CryptoAlgId ara::crypto::kAlgIdNone = kAlgIdUndefined

[SWS_CRYPT_13003]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::kAlgIdNone
Scope:	namespace ara::crypto
Type:	const CryptoAlgId
Syntax:	`const CryptoAlgId ara::crypto::kAlgIdNone= kAlgIdUndefined;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	None of Algorithm ID (i.e. an algorithm definition is not applicable).

⌋*(RS_CRYPTO_02107)*

8.8 Allowed Usage bit-flags

Variables

- const AllowedUsageFlags ara::crypto::kAllowPrototypedOnly = 0L
- const AllowedUsageFlags ara::crypto::kAllowDataEncryption = 0x0001L
- const AllowedUsageFlags ara::crypto::kAllowDataDecryption = 0x0002L
- const AllowedUsageFlags ara::crypto::kAllowSignature = 0x0004L
- const AllowedUsageFlags ara::crypto::kAllowVerification = 0x0008L
- const AllowedUsageFlags ara::crypto::kAllowKeyAgreement = 0x0010L
- const AllowedUsageFlags ara::crypto::kAllowKeyDiversify = 0x0020L
- const AllowedUsageFlags ara::crypto::kAllowDrngInit = 0x0040L
- const AllowedUsageFlags ara::crypto::kAllowKdfMaterial = 0x0080L
- const AllowedUsageFlags ara::crypto::kAllowKeyExporting = 0x0100L
- const AllowedUsageFlags ara::crypto::kAllowKeyImporting = 0x0200L
- const AllowedUsageFlags ara::crypto::kAllowExactModeOnly = 0x8000L
- const AllowedUsageFlags ara::crypto::kAllowDerivedDataEncryption = kAllowDataEncryption << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedDataDecryption = kAllowDataDecryption << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedSignature = kAllowSignature << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedVerification = kAllowVerification << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedKeyDiversify = kAllowKeyDiversify << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedDrngInit = kAllowDrngInit << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedKdfMaterial = kAllowKdfMaterial << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedKeyExporting = kAllowKeyExporting << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedKeyImporting = kAllowKeyImporting << 16
- const AllowedUsageFlags ara::crypto::kAllowDerivedExactModeOnly = kAllowExactModeOnly << 16
- const AllowedUsageFlags ara::crypto::kAllowKdfMaterialAnyUsage

Detailed Description

This group contains list of constant 1-bit values predefined for Allowed Usage flags.

8.8.1 Variable Documentation

8.8.1.1 const AllowedUsageFlags ara::crypto::kAllowPrototypedOnly = 0L

[SWS_CRYPT_13100]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowPrototypedOnly
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowPrototypedOnly= 0L;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The key/seed usage will be fully specified by a key slot prototype (the object can be used only after reloading from the slot).

⌋*(RS_CRYPTO_02111)*

8.8.1.2 const AllowedUsageFlags ara::crypto::kAllowDataEncryption = 0x0001L

[SWS_CRYPT_13101]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDataEncryption
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDataEncryption= 0x0001L;`
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)*

8.8.1.3 const AllowedUsageFlags ara::crypto::kAllowDataDecryption = 0x0002L

[SWS_CRYPT_13102]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDataDecryption
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDataDecryption= 0x0002L;`
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)*

8.8.1.4 const AllowedUsageFlags ara::crypto::kAllowSignature = 0x0004L

[SWS_CRYPT_13103]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowSignature
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowSignature= 0x0004L;`
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)*

8.8.1.5 const AllowedUsageFlags ara::crypto::kAllowVerification = 0x0008L

[SWS_CRYPT_13104]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowVerification
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowVerification= 0x0008L;`
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)*

8.8.1.6 const AllowedUsageFlags ara::crypto::kAllowKeyAgreement = 0x0010L

[SWS_CRYPT_13105]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowKeyAgreement
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowKeyAgreement= 0x0010L;`
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)*

8.8.1.7 const AllowedUsageFlags ara::crypto::kAllowKeyDiversify = 0x0020L

[SWS_CRYPT_13106]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowKeyDiversify
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowKeyDiversify= 0x0020L;`
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)*

8.8.1.8 const AllowedUsageFlags ara::crypto::kAllowDrngInit = 0x0040L

[SWS_CRYPT_13107]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDrngInit
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDrngInit= 0x0040L;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	The seed or symmetric key can be used for initialization of a Deterministic Random Number Generators (DRNG) or "mixed" implementations (DRNG + TRNG).

⌋*(RS_CRYPTO_02111)*

8.8.1.9 const AllowedUsageFlags ara::crypto::kAllowKdfMaterial = 0x0080L

[SWS_CRYPT_13108]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowKdfMaterial
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowKdfMaterial= 0x0080L;`
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 KeyMaterial for slave-keys derivation via a Key Derivation Function (KDF).

⌋*(RS_CRYPTO_02111)*

8.8.1.10 const AllowedUsageFlags ara::crypto::kAllowKeyExporting = 0x0100L

[SWS_CRYPT_13109]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowKeyExporting
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowKeyExporting= 0x0100L;`
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)*

8.8.1.11 const AllowedUsageFlags ara::crypto::kAllowKeyImporting = 0x0200L

[SWS_CRYPT_13110]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowKeyImporting
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowKeyImporting= 0x0200L;`
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)*

8.8.1.12 const AllowedUsageFlags ara::crypto::kAllowExactModeOnly = 0x8000L

[SWS_CRYPT_13111]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowExactModeOnly
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowExactModeOnly= 0x8000L;`
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)*

8.8.1.13 const AllowedUsageFlags ara::crypto::kAllowDerivedDataEncryption = kAllowDataEncryption << 16

[SWS_CRYPT_13112]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedDataEncryption
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDerivedDataEncryption= kAllowDataEncryption << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for data encryption.

⌋*(RS_CRYPTO_02111)*

8.8.1.14 const AllowedUsageFlags ara::crypto::kAllowDerivedDataDecryption = kAllowDataDecryption << 16

[SWS_CRYPT_13113]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedDataDecryption
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDerivedDataDecryption= kAllowDataDecryption << 16;`

▽

△

Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for data decryption.

⌋*(RS_CRYPTO_02111)*

8.8.1.15 const AllowedUsageFlags ara::crypto::kAllowDerivedSignature = kAllowSignature << 16

[SWS_CRYPT_13114]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedSignature
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::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)*

8.8.1.16 const AllowedUsageFlags ara::crypto::kAllowDerivedVerification = kAllowVerification << 16

[SWS_CRYPT_13115]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedVerification
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::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)*

8.8.1.17 const AllowedUsageFlags ara::crypto::kAllowDerivedKeyDiversify = kAllowKeyDiversify << 16

[SWS_CRYPT_13116]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedKeyDiversify
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDerivedKeyDiversify= kAllowKeyDiversify << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for slave-keys diversification.

⌋*(RS_CRYPTO_02111)*

8.8.1.18 const AllowedUsageFlags ara::crypto::kAllowDerivedDrngInit = kAllowDrngInit $<<$ 16

[SWS_CRYPT_13117]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedDrngInit
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDerivedDrngInit= kAllowDrngInit << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used for initialization of a Deterministic Random Number Generators (DRNG).

⌋*(RS_CRYPTO_02111)*

8.8.1.19 const AllowedUsageFlags ara::crypto::kAllowDerivedKdfMaterial = kAllowKdfMaterial $<<$ 16

[SWS_CRYPT_13118]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedKdfMaterial
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDerivedKdfMaterial= kAllowKdfMaterial << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used as a KeyMaterial for slave-keys derivation via a Key Derivation Function (KDF).

⌋*(RS_CRYPTO_02111)*

8.8.1.20 const AllowedUsageFlags ara::crypto::kAllowDerivedKeyExporting = kAllowKeyExporting << 16

[SWS_CRYPT_13119]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedKeyExporting
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDerivedKeyExporting= kAllowKeyExporting << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	A derived seed or symmetric key can be used as a "transport" one for Key-Wrap transformation.

⌋*(RS_CRYPTO_02111)*

8.8.1.21 const AllowedUsageFlags ara::crypto::kAllowDerivedKeyImporting = kAllowKeyImporting << 16

[SWS_CRYPT_13120]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedKeyImporting
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::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)*

8.8.1.22 const AllowedUsageFlags ara::crypto::kAllowDerivedExactModeOnly = kAllowExactModeOnly << 16

[SWS_CRYPT_13121]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowDerivedExactModeOnly
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags

▽

△

Syntax:	`const AllowedUsageFlags ara::crypto::kAllowDerivedExactModeOnly= kAllowExactModeOnly << 16;`
Header file:	#include "ara/crypto/common/base_id_types.h"
Description:	Restrict usage of derived objects to specified operation mode only. A derived seed or symmetric key can be used only for the mode directly specified by Key::AlgId.

⌋*(RS_CRYPTO_02111)*

8.8.1.23 const AllowedUsageFlags ara::crypto::kAllowKdfMaterialAnyUsage

Initial value:

```
= kAllowKdfMaterial | kAllowDerivedDataEncryption
| kAllowDerivedDataDecryption |
      kAllowDerivedSignature | kAllowDerivedVerification |
      kAllowDerivedKeyDiversify
| kAllowDerivedDrngInit | kAllowDerivedKdfMaterial |
      kAllowDerivedKeyExporting | kAllowDerivedKeyImporting
```

[SWS_CRYPT_13122]{DRAFT} ⌈

Kind:	variable
Symbol:	AllowedUsageConsts::kAllowKdfMaterialAnyUsage
Scope:	group AllowedUsageConsts
Type:	const AllowedUsageFlags
Syntax:	`const AllowedUsageFlags ara::crypto::kAllowKdfMaterialAnyUsage= kAllowKdfMaterial | kAllowDerivedDataEncryption | kAllowDerivedDataDecryption | kAllowDerivedSignature | kAllowDerivedVerification | kAllowDerivedKeyDiversify | kAllowDerivedDrngInit | 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 KeyMaterial for a Key Derivation Function (KDF) and the derived "slave" keys can be used without limitations.

⌋*(RS_CRYPTO_02111)*

8.9 Basic common interfaces

Classes

- class ara::crypto::CustomDisposable
- struct ara::crypto::CustomDeleter
- struct ara::crypto::ProviderInfo::Version
- class ara::crypto::ProviderInfo
- class ara::crypto::Serializable
- class ara::crypto::TrustedContainer

Functions

- bool ara::crypto::operator== (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool ara::crypto::operator< (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool ara::crypto::operator> (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool ara::crypto::operator!= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool ara::crypto::operator<= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- bool ara::crypto::operator>= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept
- std::uint64_t ara::crypto::ProviderInfo::Version::Encode () const noexcept
- bool ara::crypto::ProviderInfo::Version::IsNil () const noexcept
- static Version ara::crypto::ProviderInfo::DecodeVersionNumber (std::uint64_t versionNumber) noexcept

Detailed Description

Group of basic common interfaces shared by all functional sub-domains of CryptoStack.

8.9.1 Class Documentation

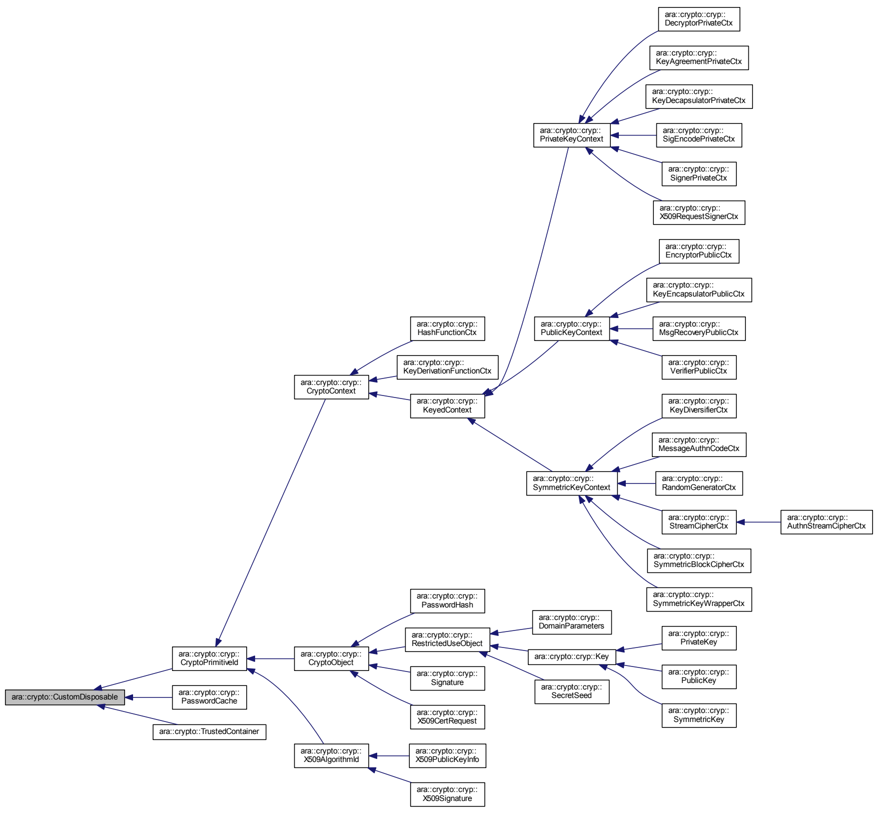
8.9.1.1 class ara::crypto::CustomDisposable

[SWS_CRYPT_10200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::CustomDisposable
Scope:	namespace ara::crypto
Syntax:	`class CustomDisposable {...};`
Header file:	#include "ara/crypto/common/custom_disposable.h"
Description:	A basic interface of customly disposable objects.

⌋*(RS_CRYPTO_02404)*

Inheritance diagram for ara::crypto::CustomDisposable:


ara::crypto::cryp:: DecryptorPrivateCtx
ara::crypto::cryp:: KeyAgreementPrivateCtx
ara::crypto::cryp:: KeyDecapsulatorPrivateCtx
ara::crypto::cryp:: PrivateKeyContext
ara::crypto::cryp:: SigEncodePrivateCtx
ara::crypto::cryp:: SignerPrivateCtx
ara::crypto::cryp:: X509RequestSignerCtx
ara::crypto::cryp:: EncryptorPublicCtx
ara::crypto::cryp:: KeyEncapsulatorPublicCtx
ara::crypto::cryp:: HashFunctionCtx
ara::crypto::cryp:: PublicKeyContext
ara::crypto::cryp:: MsgRecoveryPublicCtx
ara::crypto::cryp:: KeyDerivationFunctionCtx
ara::crypto::cryp:: VerifierPublicCtx
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: KeyDiversifierCtx
ara::crypto::cryp:: MessageAuthnCodeCtx
ara::crypto::cryp:: SymmetricKeyContext
ara::crypto::cryp:: RandomGeneratorCtx
ara::crypto::cryp:: StreamCipherCtx
ara::crypto::cryp:: AuthnStreamCipherCtx
ara::crypto::cryp:: SymmetricBlockCipherCtx
ara::crypto::cryp:: SymmetricKeyWrapperCtx
ara::crypto::cryp:: PasswordHash
ara::crypto::cryp:: DomainParameters
ara::crypto::cryp:: RestrictedUseObject
ara::crypto::cryp:: PrivateKey
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoObject
ara::crypto::cryp::Key
ara::crypto::cryp:: Signature
ara::crypto::cryp:: PublicKey
ara::crypto::CustomDisposable
ara::crypto::cryp:: PasswordCache
ara::crypto::cryp:: SecretSeed
ara::crypto::cryp:: X509CertRequest
ara::crypto::cryp:: SymmetricKey
ara::crypto::TrustedContainer
ara::crypto::cryp:: X509AlgorithmId
ara::crypto::cryp:: X509PublicKeyInfo
ara::crypto::cryp:: X509Signature

Public Member Functions

- virtual void Release () const noexcept=0

Protected Member Functions

- virtual ∼CustomDisposable () noexcept=default

8.9.1.1.1 Constructor & Destructor Documentation

8.9.1.1.1.1 virtual ara::crypto::CustomDisposable::∼CustomDisposable ( ) [protected], [virtual], [default], [noexcept]

[SWS_CRYPT_10210]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::CustomDisposable::~CustomDisposable()
Scope:	class ara::crypto::CustomDisposable
Visibility:	protected
Syntax:	`virtual ~CustomDisposable () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/custom_disposable.h"
Description:	Hides destructor from client side code.

⌋*(RS_CRYPTO_02311)*

8.9.1.1.2 Member Function Documentation

8.9.1.1.2.1 virtual void ara::crypto::CustomDisposable::Release ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_10211]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::CustomDisposable::Release()
Scope:	class ara::crypto::CustomDisposable
Syntax:	`virtual void Release () const noexcept=0;`
Return value:	None
Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/common/custom_disposable.h"
Description:	Release allocated memory and other resources.
Notes:	This method must be single mechanism to destroy any interface instance allocated by the Crypto Stack. All variants of the delete operator for this interface must be hidden from a consumer code!

⌋*(RS_CRYPTO_02404)*

8.9.1.2 struct ara::crypto::CustomDeleter

[SWS_CRYPT_10300]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::crypto::CustomDeleter
Scope:	namespace ara::crypto
Syntax:	`struct CustomDeleter {...};`
Header file:	#include "ara/crypto/common/custom_disposable.h"
Description:	A custom deleter definition.

⌋*(RS_CRYPTO_02404)*

Public Member Functions

- constexpr CustomDeleter () noexcept=default
- void operator() (const CustomDisposable ∗ptr) const noexcept

8.9.1.2.1 Constructor & Destructor Documentation

8.9.1.2.1.1 constexpr ara::crypto::CustomDeleter::CustomDeleter ( ) `[default]`, `[noexcept]`

[SWS_CRYPT_10311]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::CustomDeleter::CustomDeleter()
Scope:	struct ara::crypto::CustomDeleter
Syntax:	`constexpr CustomDeleter () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/custom_disposable.h"
Description:	Constructor of the Custom Deleter.

⌋*(RS_CRYPTO_02311)*

8.9.1.2.2 Member Function Documentation

8.9.1.2.2.1 void ara::crypto::CustomDeleter::operator() (const CustomDisposable ∗ *ptr*) const `[noexcept]`

[SWS_CRYPT_10312]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::CustomDeleter::operator()(const CustomDisposable *ptr)	
Scope:	struct ara::crypto::CustomDeleter	
Syntax:	`inline void operator() (const CustomDisposable *ptr) const noexcept;`	
Parameters (in):	ptr	the pointer to an instance of the CustomDisposable interface
Return value:	None	
Exception Safety:	noexcept	
Thread Safety:	Reentrant	
Header file:	#include "ara/crypto/common/custom_disposable.h"	
Description:	Custom Delete operator.	

⌋*(RS_CRYPTO_02311)*

8.9.1.3 struct ara::crypto::ProviderInfo::Version

[SWS_CRYPT_10600]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::crypto::ProviderInfo::Version
Scope:	class ara::crypto::ProviderInfo
Syntax:	`struct Version {...};`
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	The Provider's Version Structure.
Notes:	The major.minor.patch fields should satisfy to the Semantic Versioning 2.0.0 (https://semver.org/).

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

Public Member Functions

- std::uint64_t Encode () const noexcept
- bool IsNil () const noexcept

Public Attributes

- std::uint64_t buildTime
- std::uint_fast16_t major
- std::uint_fast16_t minor
- std::uint_fast16_t patch
- bool isRelease

8.9.1.3.1 Member Data Documentation

8.9.1.3.1.1 std::uint64_t ara::crypto::ProviderInfo::Version::buildTime

[SWS_CRYPT_10601]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::ProviderInfo::Version::buildTime
Scope:	struct ara::crypto::ProviderInfo::Version
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::crypto::ProviderInfo::Version::buildTime;`
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	Build time stamp (the number of seconds since the UNIX Epoch, 1 January 1970)

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.1.3.1.2 std::uint_fast16_t ara::crypto::ProviderInfo::Version::major

[SWS_CRYPT_10602]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::ProviderInfo::Version::major
Scope:	struct ara::crypto::ProviderInfo::Version
Type:	std::uint_fast16_t
Syntax:	`std::uint_fast16_t ara::crypto::ProviderInfo::Version::major;`
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	Major version number

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.1.3.1.3 std::uint_fast16_t ara::crypto::ProviderInfo::Version::minor

[SWS_CRYPT_10603]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::ProviderInfo::Version::minor
Scope:	struct ara::crypto::ProviderInfo::Version
Type:	std::uint_fast16_t
Syntax:	`std::uint_fast16_t ara::crypto::ProviderInfo::Version::minor;`
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	Minor version number

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.1.3.1.4 std::uint_fast16_t ara::crypto::ProviderInfo::Version::patch

[SWS_CRYPT_10604]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::ProviderInfo::Version::patch
Scope:	struct ara::crypto::ProviderInfo::Version
Type:	std::uint_fast16_t
Syntax:	`std::uint_fast16_t ara::crypto::ProviderInfo::Version::patch;`
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	Patch number

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.1.3.1.5 bool ara::crypto::ProviderInfo::Version::isRelease

[SWS_CRYPT_10605]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::ProviderInfo::Version::isRelease
Scope:	struct ara::crypto::ProviderInfo::Version
Type:	bool
Syntax:	`bool ara::crypto::ProviderInfo::Version::isRelease;`
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	Release flag: it is a "Release" version if true and "Pre-release" if false

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

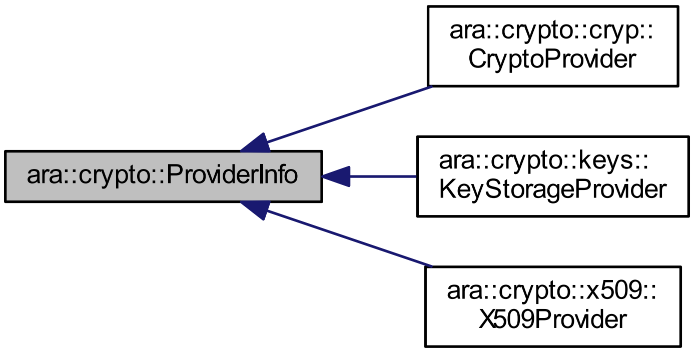
8.9.1.4 class ara::crypto::ProviderInfo

[SWS_CRYPT_10500]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::ProviderInfo
Scope:	namespace ara::crypto
Syntax:	`class ProviderInfo {...};`
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	A common interface for obtaining an identification information of a Provider.

⌋*(RS_CRYPTO_02311)*

Inheritance diagram for ara::crypto::ProviderInfo:



Public Member Functions

- virtual std::uint64_t GetProviderVersion () const noexcept=0
- virtual const char $*$ GetProviderName () const noexcept=0
- virtual void GetProviderUid (Guid &providerUid) const noexcept=0
- virtual ProviderType GetProviderType () const noexcept=0

Static Public Member Functions

- static Version DecodeVersionNumber (std::uint64_t versionNumber) noexcept

Protected Member Functions

- virtual ∼ProviderInfo () noexcept=default

8.9.1.4.1 Constructor & Destructor Documentation

8.9.1.4.1.1 virtual ara::crypto::ProviderInfo::∼ProviderInfo ( ) [protected], [virtual], [default], [noexcept]

[SWS_CRYPT_10510]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::ProviderInfo::~ProviderInfo()
Scope:	class ara::crypto::ProviderInfo
Visibility:	protected
Syntax:	`virtual ~ProviderInfo () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/provider_info.h"
Description:	Destructor.

⌋*(RS_CRYPTO_02311)*

8.9.1.4.2 Member Function Documentation

8.9.1.4.2.1 virtual std::uint64_t ara::crypto::ProviderInfo::GetProviderVersion ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10511]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::ProviderInfo::GetProviderVersion()	
Scope:	class ara::crypto::ProviderInfo	
Syntax:	`virtual std::uint64_t GetProviderVersion () const noexcept=0;`	
Return value:	std::uint64_t	encoded "version number" of the Provider (its structure is presented below)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Return an encoded "version number" of the Provider.	
Notes:	The returned "version number" can be decoded by the method DecodeVersion Number(std::uint64_t). The encoded "version numbers" can be compared directly.	

⌋*(RS_CRYPTO_02404)*

8.9.1.4.2.2 virtual const char∗ ara::crypto::ProviderInfo::GetProviderName ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10512]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::ProviderInfo::GetProviderName()	
Scope:	class ara::crypto::ProviderInfo	
Syntax:	`virtual const char* GetProviderName () const noexcept=0;`	
Return value:	const char *	a pointer to null-terminated string with Provider Name
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Get a human readable name of the Provider.	
Notes:	Life-time of the returned string is not less than the Provider instance life-time.	

⌋(*RS_CRYPTO_02311*)

8.9.1.4.2.3 virtual void ara::crypto::ProviderInfo::GetProviderUid (Guid & *providerUid*) const [pure virtual], [noexcept]

[SWS_CRYPT_10513]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::ProviderInfo::GetProviderUid(Guid &providerUid)	
Scope:	class ara::crypto::ProviderInfo	
Syntax:	`virtual void GetProviderUid (Guid &providerUid) const noexcept=0;`	
Parameters (out):	providerUid	the output buffer for the Provider's GUID
Return value:	None	
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Get the Provider's Globally Unique Identifier (GUID).	

⌋(*RS_CRYPTO_02401, RS_CRYPTO_02404*)

8.9.1.4.2.4 virtual ProviderType ara::crypto::ProviderInfo::GetProviderType ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_10514]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::ProviderInfo::GetProviderType()
Scope:	class ara::crypto::ProviderInfo
Syntax:	`virtual ProviderType GetProviderType () const noexcept=0;`

▽

△

Return value:	ProviderType	type of the Provider
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Get type of the Provider.	

⌋*(RS_CRYPTO_02109, RS_CRYPTO_02401)*

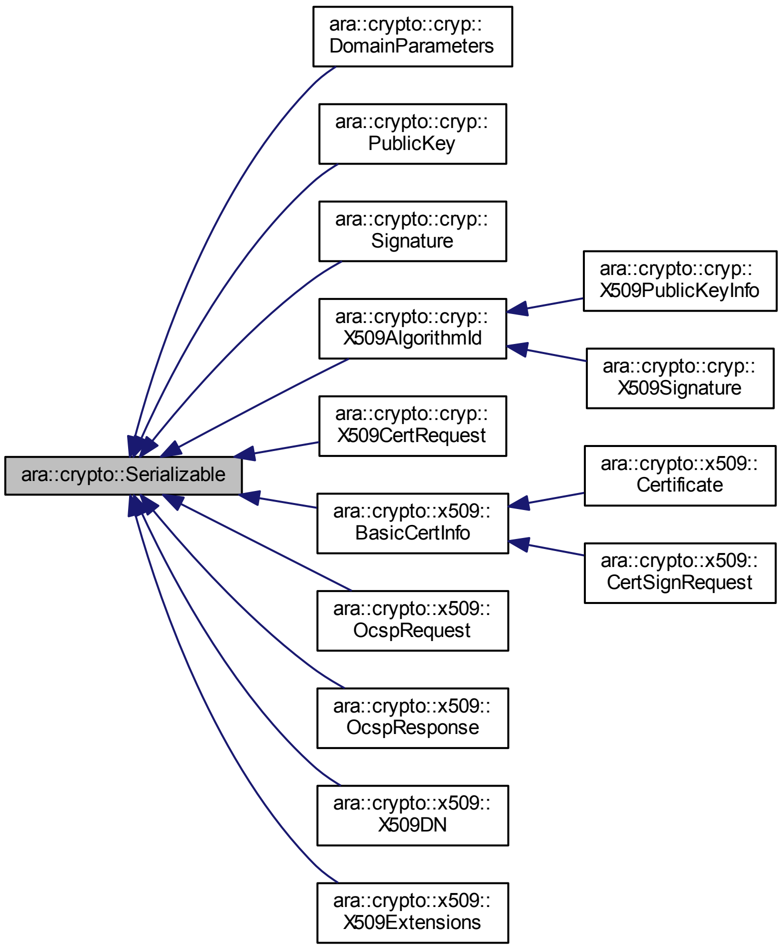
8.9.1.5 class ara::crypto::Serializable

[SWS_CRYPT_10700]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::Serializable
Scope:	namespace ara::crypto
Syntax:	`class Serializable {...};`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Serializable object interface.

⌋*(RS_CRYPTO_02105)*

Inheritance diagram for ara::crypto::Serializable:


ara::crypto::cryp::DomainParameters
ara::crypto::cryp::PublicKey
ara::crypto::cryp::Signature
ara::crypto::cryp::X509AlgorithmId
ara::crypto::cryp::X509PublicKeyInfo
ara::crypto::cryp::X509Signature
ara::crypto::cryp::X509CertRequest
ara::crypto::Serializable
ara::crypto::x509::Certificate
ara::crypto::x509::BasicCertInfo
ara::crypto::x509::CertSignRequest
ara::crypto::x509::OcspRequest
ara::crypto::x509::OcspResponse
ara::crypto::x509::X509DN
ara::crypto::x509::X509Extensions

Public Types

- using FormatId = std::uint32_t

Public Member Functions

- virtual ara::core::Result< std::size_t > ExportPublicly (WritableMemRegion output=WritableMemRegion(), FormatId formatId=kFormatDefault) const noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< void > ExportPublicly (ByteVectorT< Alloc > &output, FormatId formatId=kFormatDefault) const noexcept

Protected Member Functions

- virtual ∼Serializable () noexcept=default

Predefined encoding format identifiers

- static const FormatId kFormatDefault = 0L
- static const FormatId kFormatRawValueOnly = 1L
- static const FormatId kFormatDerEncoded = 2L
- static const FormatId kFormatPemEncoded = 3L

8.9.1.5.1 Member Typedef Documentation

8.9.1.5.1.1 using ara::crypto::Serializable::FormatId = std::uint32_t

[SWS_CRYPT_10701]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::Serializable::FormatId
Scope:	class ara::crypto::Serializable
Derived from:	std::uint32_t
Syntax:	`using ara::crypto::Serializable::FormatId = std::uint32_t;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	A container type for the encoding format identifiers.

⌋*(RS_CRYPTO_02311)*

8.9.1.5.2 Constructor & Destructor Documentation

8.9.1.5.2.1 virtual ara::crypto::Serializable::∼Serializable ( ) `[protected]`, `[virtual]`, `[default]`, `[noexcept]`

[SWS_CRYPT_10710]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::Serializable::~Serializable()
Scope:	class ara::crypto::Serializable
Visibility:	protected

▽

△

Syntax:	`virtual ~Serializable () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Destructor.

⌋(*RS_CRYPTO_02311*)

8.9.1.5.3 Member Function Documentation

8.9.1.5.3.1 virtual ara::core::Result<std::size_t> ara::crypto::Serializable::ExportPublicly (WritableMemRegion *output* = WritableMemRegion (), FormatId *formatId* = kFormatDefault) const [pure virtual], [noexcept]

[SWS_CRYPT_10711]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::Serializable::ExportPublicly(WritableMemRegion output=WritableMemRegion()	
Scope:	class ara::crypto::Serializable	
Syntax:	`virtual ara::core::Result<std::size_t> ExportPublicly (WritableMem Region output=WritableMemRegion(), FormatId formatId=kFormatDefault) const noexcept=0;`	
Parameters (in):	formatId	the Crypto Provider specific identifier of the output format
Parameters (out):	output	the preallocated output buffer (it can be empty if only the required size of the output buffer is interested)
Return value:	ara::core::Result< std::size_t >	size required for storing of the output object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/serializable.h"	
Description:	Serialize itself publicly.	
Notes:	[Error]: SecurityErrorDomain::kInsufficientCapacity if (output.empty() == false), but it's capacity is less than required [Error]: SecurityErrorDomain::kUnknownIdentifier if an unknown format ID was specified [Error]: SecurityErrorDomain::kUnsupportedFormat if the specified format ID is not supported for this object type	

⌋(*RS_CRYPTO_02112*)

8.9.1.5.3.2 template<typename Alloc = DefBytesAllocator> ara::core::Result<void> ara::crypto::Serializable::ExportPublicly (ByteVectorT< Alloc > & *output*, FormatId *formatId* = kFormatDefault) const [noexcept]

[SWS_CRYPT_10712]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::Serializable::ExportPublicly(ByteVectorT< Alloc > &output, FormatId formatId=kFormatDefault)	
Scope:	class ara::crypto::Serializable	
Syntax:	`template <typename Alloc> inline ara::core::Result<void> ExportPublicly (ByteVectorT< Alloc > &output, 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
Parameters (out):	output	pre-reserved managed container for the serialization output
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/serializable.h"	
Description:	Serialize itself publicly.	
Notes:	This method sets the size of the output container according to actually saved value! [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the output buffer is less than required [Error]: SecurityErrorDomain::kUnknownIdentifier if an unknown format ID was specified [Error]: SecurityErrorDomain::kUnsupportedFormat if the specified format ID is not supported for this object type	

⌋*(RS_CRYPTO_02112)*

8.9.1.5.4 Member Data Documentation

8.9.1.5.4.1 const FormatId ara::crypto::Serializable::kFormatDefault = 0L `[static]`

[SWS_CRYPT_10750]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::Serializable::kFormatDefault
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId ara::crypto::Serializable::kFormatDefault= 0L;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Default serialization format.

⌋*(RS_CRYPTO_02311)*

8.9.1.5.4.2 const FormatId ara::crypto::Serializable::kFormatRawValueOnly = 1L `[static]`

[SWS_CRYPT_10751]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::Serializable::kFormatRawValueOnly
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId ara::crypto::Serializable::kFormatRawValueOnly= 1L;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Export only raw value of an object.

⌋*(RS_CRYPTO_02311)*

8.9.1.5.4.3 const FormatId ara::crypto::Serializable::kFormatDerEncoded = 2L `[static]`

[SWS_CRYPT_10752]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::Serializable::kFormatDerEncoded
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId ara::crypto::Serializable::kFormatDerEncoded= 2L;`
Header file:	#include "ara/crypto/common/serializable.h"
Description:	Export DER-encoded value of an object.

⌋*(RS_CRYPTO_02311)*

8.9.1.5.4.4 const FormatId ara::crypto::Serializable::kFormatPemEncoded = 3L `[static]`

[SWS_CRYPT_10753]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::Serializable::kFormatPemEncoded
Scope:	class ara::crypto::Serializable
Type:	const FormatId
Syntax:	`const FormatId ara::crypto::Serializable::kFormatPemEncoded= 3L;`
Header file:	#include "ara/crypto/common/serializable.h"

▽

△

Description:	Export PEM-encoded value of an object.

⌋(*RS_CRYPTO_02311*)

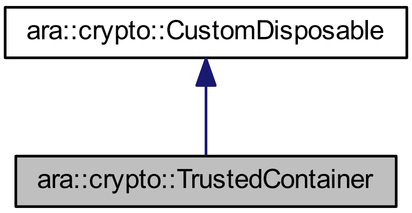
8.9.1.6 class ara::crypto::TrustedContainer

[SWS_CRYPT_10800]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::TrustedContainer
Scope:	namespace ara::crypto
Base class:	ara::crypto::CustomDisposable
Syntax:	`class TrustedContainer :  public CustomDisposable {...};`
Header file:	#include "ara/crypto/common/trusted_container.h"
Description:	Formal interface of a Trusted Container is used for saving and loading of security objects.
Notes:	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 saving or transferring.

⌋(*RS_CRYPTO_02004*)

Inheritance diagram for ara::crypto::TrustedContainer:



Public Types

- using Uptr = std::unique_ptr< TrustedContainer, CustomDeleter >
- using Uptrc = std::unique_ptr< const TrustedContainer, CustomDeleter >
- using ContentType = CryptoObjectType

Public Member Functions

- virtual ContentType GetObjectId (CryptoObjectUid ∗objectUid=nullptr) const noexcept=0
- virtual ContentType GetDependenceId (CryptoObjectUid ∗objectUid=nullptr) const noexcept=0
- virtual std::size_t Capacity () const noexcept=0
- virtual bool IsVolatile () const noexcept=0
- virtual bool IsObjectSession () const noexcept=0
- virtual bool IsObjectExportable () const noexcept=0
- virtual std::size_t ObjectSize () const noexcept=0
- virtual ContentType TypeRestriction () const noexcept=0
- virtual AllowedUsageFlags AllowedUsage () const noexcept=0
- virtual std::size_t GetReferencesCounter () const noexcept=0
- virtual bool HasOwnership () const noexcept=0

Protected Member Functions

- virtual ∼TrustedContainer () noexcept=default

8.9.1.6.1 Member Typedef Documentation

8.9.1.6.1.1 using ara::crypto::TrustedContainer::Uptr = std::unique_ptr<TrustedContainer, CustomDeleter>

[SWS_CRYPT_10801]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::TrustedContainer::Uptr
Scope:	class ara::crypto::TrustedContainer
Derived from:	std::unique_ptr<TrustedContainer, CustomDeleter>
Syntax:	`using ara::crypto::TrustedContainer::Uptr = std::unique_ptr<TrustedContainer, CustomDeleter>;`
Header file:	#include "ara/crypto/common/trusted_container.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.9.1.6.1.2 using ara::crypto::TrustedContainer::Uptrc = std::unique_ptr<const TrustedContainer, CustomDeleter>

[SWS_CRYPT_10802]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::TrustedContainer::Uptrc
Scope:	class ara::crypto::TrustedContainer
Derived from:	std::unique_ptr<const TrustedContainer, CustomDeleter>
Syntax:	`using ara::crypto::TrustedContainer::Uptrc = std::unique_ptr<const TrustedContainer, CustomDeleter>;`
Header file:	#include "ara/crypto/common/trusted_container.h"
Description:	Unique smart pointer of the constant interface.

⌋*(RS_CRYPTO_02404)*

8.9.1.6.1.3 using ara::crypto::TrustedContainer::ContentType = CryptoObjectType

[SWS_CRYPT_10803]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::TrustedContainer::ContentType
Scope:	class ara::crypto::TrustedContainer
Derived from:	CryptoObjectType
Syntax:	`using ara::crypto::TrustedContainer::ContentType = CryptoObjectType;`
Header file:	#include "ara/crypto/common/trusted_container.h"
Description:	Content Type of the Trusted Container.

⌋*(RS_CRYPTO_02004)*

8.9.1.6.2 Constructor & Destructor Documentation

8.9.1.6.2.1 virtual ara::crypto::TrustedContainer::∼TrustedContainer ( ) `[protected]`, `[virtual]`, `[default]`, `[noexcept]`

[SWS_CRYPT_10810]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::TrustedContainer::~TrustedContainer()
Scope:	class ara::crypto::TrustedContainer

▽

△

Visibility:	protected
Syntax:	`virtual ~TrustedContainer () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/common/trusted_container.h"
Description:	Destructor.

⌋*(RS_CRYPTO_02311)*

8.9.1.6.3 Member Function Documentation

8.9.1.6.3.1 virtual ContentType ara::crypto::TrustedContainer::GetObjectId (CryptoObjectUid ∗ *objectUid* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10811]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::GetObjectId(CryptoObjectUid *objectUid=nullptr)	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual ContentType GetObjectId (CryptoObjectUid *objectUid=nullptr) const noexcept=0;`	
Parameters (out):	objectUid	the optional pointer to buffer for getting COUID of an object stored in the container
Return value:	ContentType	type of the content stored in the container
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return COUID and type of an object stored to this trusted container.	
Notes:	If the container is empty then this method returns ContentType::kNone. If (objectUid != nullptr), but the container is empty then the objectUid will be filled by all zeros. Unambiguous identification of a crypto object requires both components: CryptoObjectUid and ContentType. A caller code may omit the optional argument if only the content type is interested.	

⌋*(RS_CRYPTO_02004)*

8.9.1.6.3.2 virtual ContentType ara::crypto::TrustedContainer::GetDependenceId (CryptoObjectUid ∗ *objectUid* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10812]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::GetDependenceId(CryptoObjectUid *objectUid=nullptr)	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual ContentType GetDependenceId (CryptoObjectUid *object Uid=nullptr) const noexcept=0;`	
Parameters (out):	objectUid	the optional pointer to a buffer for getting COUID of an object from which current object depends
Return value:	ContentType	type of an object from which current one depends or ContentType::kNone if it has no any dependencies
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return COUID and type of an object from which current object (in the container) depends.	
Notes:	If the object doesn't depend from (or refer to) another object then the objectId will be filled by all zeros. Unambiguous identification of a crypto object requires both components: CryptoObjectUid and ContentType. A caller code may omit the optional argument if only the dependency type is interested.	

⌋*(RS_CRYPTO_02311)*

8.9.1.6.3.3 virtual std::size_t ara::crypto::TrustedContainer::Capacity ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10813]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::Capacity()	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual std::size_t Capacity () const noexcept=0;`	
Return value:	std::size_t	capacity of the trusted container (in bytes)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return capacity of the trusted container.	

⌋*(RS_CRYPTO_02110)*

8.9.1.6.3.4 virtual bool ara::crypto::TrustedContainer::IsVolatile ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10814]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::IsVolatile()	
Scope:	class ara::crypto::TrustedContainer	
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/trusted_container.h"	
Description:	Return volatility of the trusted container.	
Notes:	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_02311)*

8.9.1.6.3.5 virtual bool ara::crypto::TrustedContainer::IsObjectSession ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_10815]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::IsObjectSession()	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual bool IsObjectSession () const noexcept=0;`	
Return value:	bool	true if an object stored to the container has set the "session" attribute
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return the "session" (or "temporary") attribute of an object stored to the container.	
Notes:	A "session" object can be stored to a "volatile" container only! If a "volatile" container keeps a non-session object then it can be saved permanently.	

⌋*(RS_CRYPTO_02311)*

8.9.1.6.3.6 virtual bool ara::crypto::TrustedContainer::IsObjectExportable ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_10816]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::IsObjectExportable()	
Scope:	class ara::crypto::TrustedContainer	
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/trusted_container.h"	
Description:	Return the "exportable" attribute of an object stored to the container.	
Notes:	The exportability of an object doesn't depend from the volatility of its container.	

⌋*(RS_CRYPTO_02311)*

8.9.1.6.3.7 virtual std::size_t ara::crypto::TrustedContainer::ObjectSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_10817]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::ObjectSize()	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual std::size_t ObjectSize () const noexcept=0;`	
Return value:	std::size_t	size of an object stored to the trusted container (in bytes)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return size of an object stored to the trusted container.	
Notes:	If the container is empty then this method returns 0.	

⌋*(RS_CRYPTO_02311)*

8.9.1.6.3.8 virtual ContentType ara::crypto::TrustedContainer::TypeRestriction ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_10818]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::TypeRestriction()	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual ContentType TypeRestriction () const noexcept=0;`	
Return value:	ContentType	an object type of allowed content (ContentType::k None means without restriction)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return content type restriction of this trusted container.	
Notes:	If a container has a type restriction different from ContentType::kNone 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)*

8.9.1.6.3.9 virtual AllowedUsageFlags ara::crypto::TrustedContainer::AllowedUsage ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10819]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::AllowedUsage()	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual AllowedUsageFlags AllowedUsage () const noexcept=0;`	
Return value:	AllowedUsageFlags	allowed key/seed usage flags
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return actual allowed key/seed usage flags defined by the key slot prototype for this "Actor" and current content of the container.	
Notes:	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)*

8.9.1.6.3.10 virtual std::size_t ara::crypto::TrustedContainer::GetReferencesCounter ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10820]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::GetReferencesCounter()	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual std::size_t GetReferencesCounter () const noexcept=0;`	
Return value:	std::size_t	references counter to an object stored in the container
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Return current number of external references to a crypto object kept in the container.	

⌋*(RS_CRYPTO_02004)*

8.9.1.6.3.11 virtual bool ara::crypto::TrustedContainer::HasOwnership ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_10821]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::TrustedContainer::HasOwnership()	
Scope:	class ara::crypto::TrustedContainer	
Syntax:	`virtual bool HasOwnership () const noexcept=0;`	
Return value:	bool	true if the container is owned by this process (always true for volatile containers), and false otherwise (the current process has only User rights on the container)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/trusted_container.h"	
Description:	Check the ownership status of the current process on this trusted container.	
Notes:	A saving operation to the container can be done only if (IsReadOnly() == false)!	

⌋*(RS_CRYPTO_02004)*

8.9.2 Function Documentation

8.9.2.1 bool ara::crypto::operator== (const ProviderInfo::Version & *lhs*, const ProviderInfo::Version & *rhs*) `[noexcept]`

[SWS_CRYPT_10550]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator==(const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline bool operator== (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if the version value of lhs is equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Comparison operator "equal" for ProviderInfo::Version operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.2.2 bool ara::crypto::operator< (const ProviderInfo::Version & *lhs*, const ProviderInfo::Version & *rhs*) `[noexcept]`

[SWS_CRYPT_10551]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator<(const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline bool operator< (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if the version value of lhs is less than rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Comparison operator "less than" for ProviderInfo::Version operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.2.3 bool ara::crypto::operator> (const ProviderInfo::Version & *lhs*, const ProviderInfo::Version & *rhs*) `[noexcept]`

[SWS_CRYPT_10552]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator>(const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline bool operator> (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if the version value of lhs is greater than rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Comparison operator "greater than" for ProviderInfo::Version operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.2.4 bool ara::crypto::operator!= (const ProviderInfo::Version & *lhs*, const ProviderInfo::Version & *rhs*) `[noexcept]`

[SWS_CRYPT_10553]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator!=(const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline bool operator!= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if the version value of lhs is not equal to rhs, and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Comparison operator "not equal" for ProviderInfo::Version operands.	

⌋*(RS_CRYPTO_02311, RS_CRYPTO_02404)*

8.9.2.5 bool ara::crypto::operator<= (const ProviderInfo::Version & *lhs*, const ProviderInfo::Version & *rhs*) `[noexcept]`

[SWS_CRYPT_10554]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator<=(const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline bool operator<= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if the version value 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/provider_info.h"	
Description:	Comparison operator "less than or equal" for ProviderInfo::Version operands.	

⌋(*RS_CRYPTO_02311*, *RS_CRYPTO_02404*)

8.9.2.6 bool ara::crypto::operator>= (const ProviderInfo::Version & *lhs*, const ProviderInfo::Version & *rhs*) `[noexcept]`

[SWS_CRYPT_10555]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::operator>=(const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs)	
Scope:	namespace ara::crypto	
Syntax:	`inline bool operator>= (const ProviderInfo::Version &lhs, const ProviderInfo::Version &rhs) noexcept;`	
Parameters (in):	lhs	left-hand side operand
	rhs	right-hand side operand
Return value:	bool	true if the version value 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/provider_info.h"	
Description:	Comparison operator "greater than or equal" for ProviderInfo::Version operands.	

⌋(*RS_CRYPTO_02311*, *RS_CRYPTO_02404*)

8.9.2.7 std::uint64_t ara::crypto::ProviderInfo::Version::Encode ( ) const `[noexcept]`

[SWS_CRYPT_10611]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::ProviderInfo::Version::Encode()	
Scope:	struct ara::crypto::ProviderInfo::Version	
Syntax:	`inline std::uint64_t Encode () const noexcept;`	
Return value:	std::uint64_t	64-bit encoded version number correspondent to the structure content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Encode the Provider's Version to a single 64-bit unsigned integer.	

⌋*(RS_CRYPTO_02404)*

8.9.2.8 bool ara::crypto::ProviderInfo::Version::IsNil ( ) const `[noexcept]`

[SWS_CRYPT_10612]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::ProviderInfo::Version::IsNil()	
Scope:	struct ara::crypto::ProviderInfo::Version	
Syntax:	`inline bool IsNil () const noexcept;`	
Return value:	bool	true if the version is "nil" and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Check whether the Provider's Version is "nil" (i.e. undefined).	
Notes:	Any valid version at least must have (buildTime > 0).	

⌋*(RS_CRYPTO_02404)*

8.9.2.9 ProviderInfo::Version ara::crypto::ProviderInfo::DecodeVersionNumber (std::uint64_t *versionNumber*) `[static]`, `[noexcept]`

[SWS_CRYPT_10515]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::ProviderInfo::DecodeVersionNumber(std::uint64_t versionNumber)	
Scope:	class ara::crypto::ProviderInfo	
Syntax:	`inline ProviderInfo::Version DecodeVersionNumber (std::uint64_t versionNumber) noexcept;`	
Parameters (in):	versionNumber	the encoded "version number" of the Provider

▽

△

Return value:	Version	parsed structure of the Provider's version
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/common/provider_info.h"	
Description:	Decode the encoded "version number" of the Provider.	

⌋*(RS_CRYPTO_02404)*

8.10 Exceptions & Error Codes

Classes

- class ara::crypto::SecurityException
- class ara::crypto::ResourceException
- class ara::crypto::BadAllocException
- class ara::crypto::LogicException
- class ara::crypto::InvalidArgumentException
- class ara::crypto::UnsupportedException
- class ara::crypto::InvalidUsageOrderException
- class ara::crypto::RuntimeException
- class ara::crypto::UnexpectedValueException
- class ara::crypto::BadObjectTypeException
- class ara::crypto::UsageViolationException
- class ara::crypto::AccessViolationException
- class ara::crypto::SecurityErrorDomain

Enumerations

Functions

- constexpr ara::core::ErrorDomain const & ara::crypto::GetSecurityErrorDomain ()
- constexpr ara::core::ErrorCode ara::crypto::MakeErrorCode (SecurityErrorDomain::Errc code, ara::core::ErrorDomain::SupportDataType data, char const ∗message=nullptr)

Detailed Description

8.10.1 Class Documentation

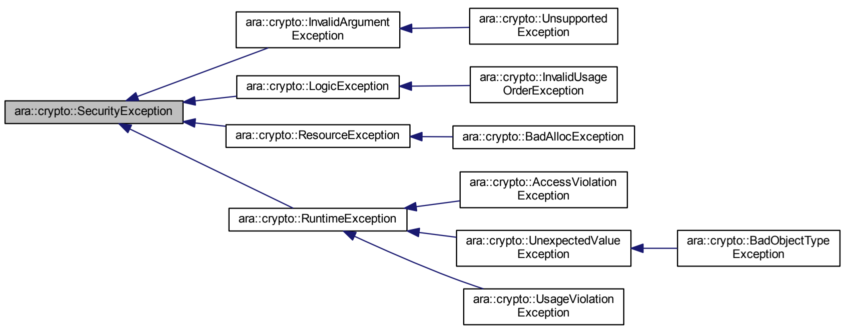
8.10.1.1 class ara::crypto::SecurityException

[SWS_CRYPT_16000]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::SecurityException
Scope:	namespace ara::crypto
Base class:	ara::core::Exception
Syntax:	`class SecurityException : public Exception {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of a Security exception.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::SecurityException:


ara::crypto::SecurityException
ara::crypto::InvalidArgument Exception
ara::crypto::Unsupported Exception
ara::crypto::LogicException
ara::crypto::InvalidUsage OrderException
ara::crypto::ResourceException
ara::crypto::BadAllocException
ara::crypto::AccessViolation Exception
ara::crypto::RuntimeException
ara::crypto::UnexpectedValue Exception
ara::crypto::BadObjectType Exception
ara::crypto::UsageViolation Exception

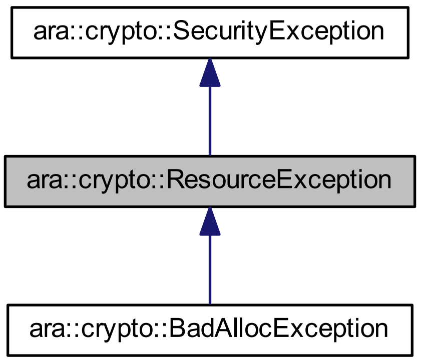
8.10.1.2 class ara::crypto::ResourceException

[SWS_CRYPT_16100]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::ResourceException
Scope:	namespace ara::crypto
Base class:	ara::crypto::SecurityException
Syntax:	`class ResourceException : public SecurityException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of a Resource fault exception.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::ResourceException:


ara::crypto::SecurityException
ara::crypto::ResourceException
ara::crypto::BadAllocException

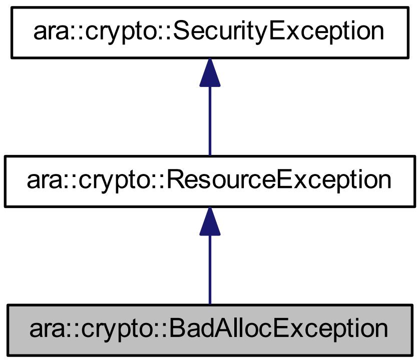
8.10.1.3 class ara::crypto::BadAllocException

[SWS_CRYPT_16200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::BadAllocException
Scope:	namespace ara::crypto
Base class:	ara::crypto::ResourceException
Syntax:	`class BadAllocException : public ResourceException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of the Bad Allocation exception.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::BadAllocException:


ara::crypto::SecurityException
ara::crypto::ResourceException
ara::crypto::BadAllocException

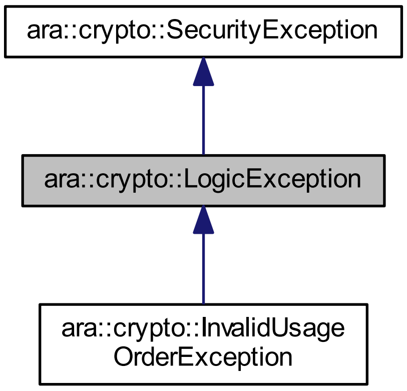
8.10.1.4 class ara::crypto::LogicException

[SWS_CRYPT_16300]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::LogicException
Scope:	namespace ara::crypto
Base class:	ara::crypto::SecurityException
Syntax:	`class LogicException : public SecurityException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of a general Security Logic Error exception. The CryptoStack should throw this exception if the incorrectness of the API call must be obvious for the consumer code even before the call execution.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::LogicException:


ara::crypto::SecurityException
ara::crypto::LogicException
ara::crypto::InvalidUsage OrderException

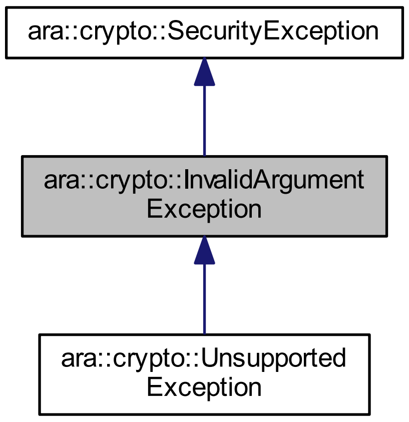
8.10.1.5 class ara::crypto::InvalidArgumentException

[SWS_CRYPT_16400]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::InvalidArgumentException
Scope:	namespace ara::crypto
Base class:	ara::crypto::SecurityException
Syntax:	`class InvalidArgumentException : public SecurityException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of the Invalid Argument exception. The CryptoStack should throw this exception if a consumer code passes to a method some invalid arguments, and their incorrectness can be detected at compile time.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::InvalidArgumentException:


ara::crypto::SecurityException
ara::crypto::InvalidArgument Exception
ara::crypto::Unsupported Exception

Public Member Functions

- std::uint8_t GetBadArgumentIndex () const noexcept

8.10.1.5.1 Member Function Documentation

8.10.1.5.1.1 std::uint8_t ara::crypto::InvalidArgumentException::GetBadArgumentIndex ( ) const [noexcept]

[SWS_CRYPT_16411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::InvalidArgumentException::GetBadArgumentIndex()	
Scope:	class ara::crypto::InvalidArgumentException	
Syntax:	`inline std::uint8_t GetBadArgumentIndex () const noexcept;`	
Return value:	std::uint8_t	1-based index of the invalid argument (0 is reserved for implicit this pointer)
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/exceptions.h"	
Description:	Get index of the Invalid Argument.	

⌋*(RS_CRYPTO_02310)*

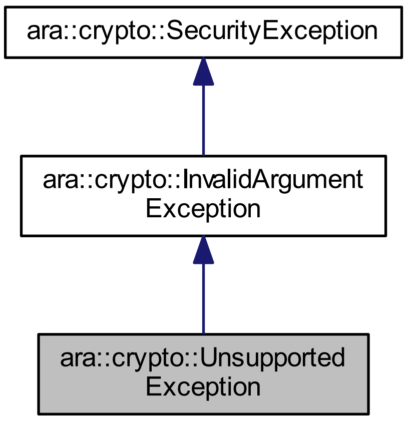
8.10.1.6 class ara::crypto::UnsupportedException

[SWS_CRYPT_16500]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::UnsupportedException
Scope:	namespace ara::crypto
Base class:	ara::crypto::InvalidArgumentException
Syntax:	`class UnsupportedException : public InvalidArgumentException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of the Crypto Unsupported method/argument exception. A Crypto Provider may have partial support of some specific algorithms or transformations and don't implement support of specific use-cases, some optional arguments or even supplementary methods. But all such restrictions should be carefully documented (in the Crypto Provider's manual) and brought to the developer attention! In a case when an application calls such unsupported API the Crypto Provider must throw this exception.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::UnsupportedException:


ara::crypto::SecurityException
ara::crypto::InvalidArgument Exception
ara::crypto::Unsupported Exception

Additional Inherited Members

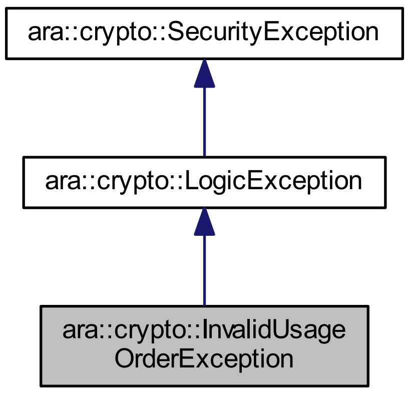
8.10.1.7 class ara::crypto::InvalidUsageOrderException

[SWS_CRYPT_16600]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::InvalidUsageOrderException
Scope:	namespace ara::crypto
Base class:	ara::crypto::LogicException
Syntax:	`class InvalidUsageOrderException : public LogicException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of a general Security Logic Error exception. The CryptoStack should throw this exception if the incorrectness of an API call can be detected at compile time.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::InvalidUsageOrderException:


ara::crypto::SecurityException
ara::crypto::LogicException
ara::crypto::InvalidUsage OrderException

8.10.1.8 class ara::crypto::RuntimeException

[SWS_CRYPT_16700]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::RuntimeException
Scope:	namespace ara::crypto
Base class:	ara::crypto::SecurityException
Syntax:	`class RuntimeException : public SecurityException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"

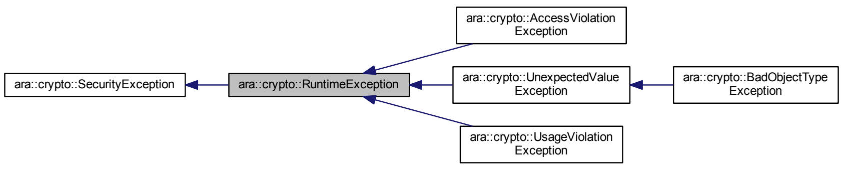
▽

△

Description:	An interface of a general Security Runtime Error exception. The CryptoStack should throw this exception if the incorrectness of an API call can be detected at runtime only.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::RuntimeException:


ara::crypto::SecurityException
ara::crypto::RuntimeException
ara::crypto::AccessViolation Exception
ara::crypto::UnexpectedValue Exception
ara::crypto::UsageViolation Exception
ara::crypto::BadObjectType Exception

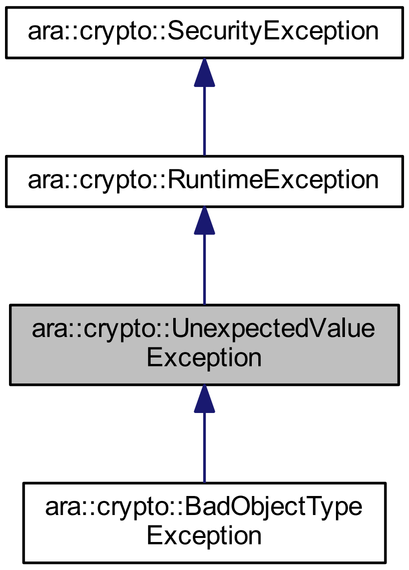
8.10.1.9 class ara::crypto::UnexpectedValueException

[SWS_CRYPT_16800]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::UnexpectedValueException
Scope:	namespace ara::crypto
Base class:	ara::crypto::RuntimeException
Syntax:	`class UnexpectedValueException : public RuntimeException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of the Unexpected Value exception. The CryptoStack should throw this exception if a consumer code pass to a method some non-expected values, but their incorrectness can be detected at runtime only.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::UnexpectedValueException:


ara::crypto::SecurityException
ara::crypto::RuntimeException
ara::crypto::UnexpectedValue Exception
ara::crypto::BadObjectType Exception

Public Member Functions

- std::uint8_t GetBadArgumentIndex () const noexcept

8.10.1.9.1 Member Function Documentation

8.10.1.9.1.1 std::uint8_t ara::crypto::UnexpectedValueException::GetBadArgumentIndex ( ) const [noexcept]

[SWS_CRYPT_16811]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::UnexpectedValueException::GetBadArgumentIndex()	
Scope:	class ara::crypto::UnexpectedValueException	
Syntax:	`inline std::uint8_t GetBadArgumentIndex () const noexcept;`	
Return value:	std::uint8_t	1-based index of the Unexpected Value argument (0 is reserved for implicit this pointer)
Exception Safety:	noexcept	

⌋

△

Header file:	#include "ara/crypto/common/exceptions.h"
Description:	Get index of the argument with the Unexpected Value.

⌋*(RS_CRYPTO_02310)*

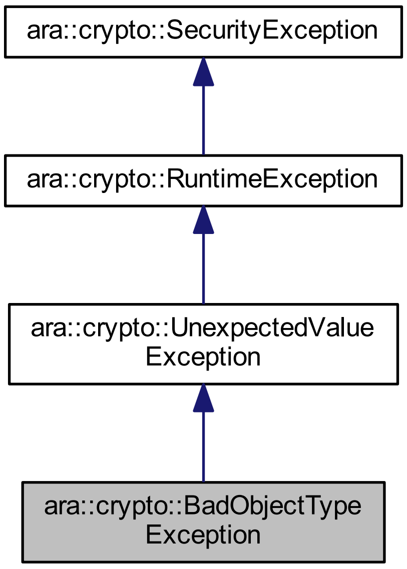
8.10.1.10 class ara::crypto::BadObjectTypeException

[SWS_CRYPT_16900]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::BadObjectTypeException
Scope:	namespace ara::crypto
Base class:	ara::crypto::UnexpectedValueException
Syntax:	`class BadObjectTypeException : public UnexpectedValueException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	Class of the Bad Crypto Object Cast exceptions. A method must throw this exception when application needs (expects) to get one type of a crypto object, but actually another type is delivered by a method call.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::BadObjectTypeException:


ara::crypto::SecurityException
ara::crypto::RuntimeException
ara::crypto::UnexpectedValue Exception
ara::crypto::BadObjectType Exception

Public Member Functions

- CryptoObjectType GetNeededType () const noexcept
- CryptoObjectType GetActualType () const noexcept

8.10.1.10.1 Member Function Documentation

8.10.1.10.1.1 CryptoObjectType ara::crypto::BadObjectTypeException::GetNeededType ( ) const [noexcept]

[SWS_CRYPT_16911]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::BadObjectTypeException::GetNeededType()	
Scope:	class ara::crypto::BadObjectTypeException	
Syntax:	`inline CryptoObjectType GetNeededType () const noexcept;`	
Return value:	CryptoObjectType	Crypto Object type expected in the operation reported this error
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/exceptions.h"	
Description:	Get the needed/expected object type in an operation throwed up this exception.	

⌋*(RS_CRYPTO_02310)*

8.10.1.10.1.2 CryptoObjectType ara::crypto::BadObjectTypeException::GetActualType ( ) const [noexcept]

[SWS_CRYPT_16912]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::BadObjectTypeException::GetActualType()	
Scope:	class ara::crypto::BadObjectTypeException	
Syntax:	`inline CryptoObjectType GetActualType () const noexcept;`	
Return value:	CryptoObjectType	actual Crypto Object type in the operation reported this error
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/exceptions.h"	
Description:	Get the actual object type in the operation throwed up this exception.	

⌋*(RS_CRYPTO_02310)*

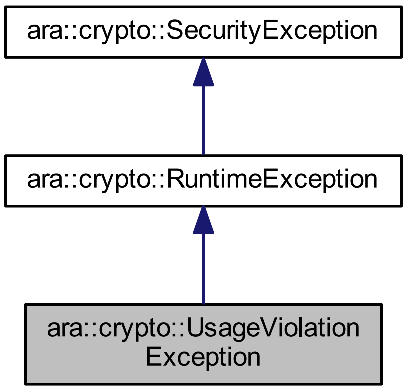
8.10.1.11 class ara::crypto::UsageViolationException

[SWS_CRYPT_17000]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::UsageViolationException
Scope:	namespace ara::crypto
Base class:	ara::crypto::RuntimeException
Syntax:	`class UsageViolationException : public RuntimeException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	An interface of the Cryptography Usage Violation exceptions. A Crypto Provider must throw this exception when application tries to violate the usage restrictions assigned to a Crypto Object. For more details see AllowedUsageFlags and related constants.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::UsageViolationException:


ara::crypto::SecurityException
ara::crypto::RuntimeException
ara::crypto::UsageViolation Exception

Public Member Functions

- AllowedUsageFlags GetAllowedUsage () const noexcept

8.10.1.11.1 Member Function Documentation

8.10.1.11.1.1 AllowedUsageFlags ara::crypto::UsageViolationException:: GetAllowedUsage ( ) const `[noexcept]`

[SWS_CRYPT_17011]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::UsageViolationException::GetAllowedUsage()	
Scope:	class ara::crypto::UsageViolationException	
Syntax:	`inline AllowedUsageFlags GetAllowedUsage () const noexcept;`	
Return value:	AllowedUsageFlags	actually allowed usage flags of the target object
Exception Safety:	noexcept	
Header file:	#include "ara/crypto/common/exceptions.h"	
Description:	Get actual "allowed usage flags" of the object (provided as an argument to the call) granted to this Actor (application/process)	

⌋*(RS_CRYPTO_02310)*

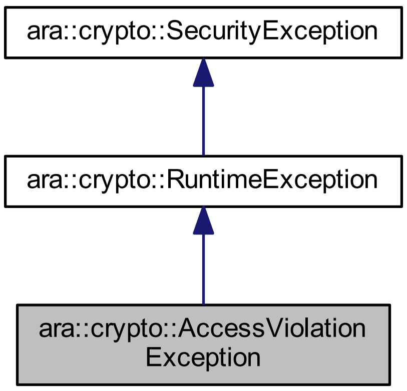
8.10.1.12 class ara::crypto::AccessViolationException

[SWS_CRYPT_17100]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::AccessViolationException
Scope:	namespace ara::crypto
Base class:	ara::crypto::RuntimeException
Syntax:	`class AccessViolationException : public RuntimeException {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	Security Access Violation exception class. The Key Storage Provider must throw this exception when an application tries violate access rights assigned to a key slot.
Notes:	This exception is specific for the Key Storage only.

⌋*(RS_CRYPTO_02310)*

Inheritance diagram for ara::crypto::AccessViolationException:


ara::crypto::SecurityException
ara::crypto::RuntimeException
ara::crypto::AccessViolation Exception

8.10.1.13 class ara::crypto::SecurityErrorDomain

[SWS_CRYPT_19900]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::SecurityErrorDomain
Scope:	namespace ara::crypto
Base class:	ara::core::ErrorDomain
Syntax:	`class SecurityErrorDomain : public ErrorDomain {...};`
Header file:	#include "ara/crypto/common/exceptions.h"
Description:	Security Error Domain class. This class represents an error domain responsible for all errors/exceptions that may be reported by public APIs in ara::crypto namespace.

⌋*(RS_CRYPTO_02310)*

Inherits ErrorDomain.

8.10.2 Enumeration Type Documentation

8.10.2.1 enum ara::crypto::SecurityErrc : ara::core::ErrorDomain::CodeType `[strong]`

[SWS_CRYPT_10099]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::SecurityErrc	
Scope:	namespace ara::crypto	
Values:	kNoError= 0	No error.
	ErrorClass= 0x1000000	Reserved (a multiplier of error class IDs)
	ErrorSubClass= 0x10000	Reserved (a multiplier of error sub-class IDs)
	ErrorSubSubClass= 0x100	Reserved (a multiplier of error sub-sub-class IDs)
	kResourceFault= 1 * ErrorClass	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!
	kBadAlloc= kResourceFault + 1 * Error SubClass	BadAllocException: Cannot allocate requested resources!
	kLogicFault= 2 * ErrorClass	LogicException: Generic logic fault!
	kInvalidArgument= kLogicFault + 1 * 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!
	kUnsupported= kInvalidArgument + 1 * ErrorSubSubClass	UnsupportedException: Unsupported request (due to limitations of the implementation)!
	kInvalidUsageOrder= kLogicFault + 2 * ErrorSubClass	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 * ErrorClass	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!
	kLockedByReference= kRuntimeFault + 5	RuntimeException: An object stored in the container is locked due to a reference from another one!
	kContentDuplication= kRuntimeFault + 6	RuntimeException: Provided content already exists in the target storage!
	kUnexpectedValue= kRuntimeFault + 1 * ErrorSubClass	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!
	kBadObjectType= kUnexpectedValue + 1 * ErrorSubSubClass	BadObjectTypeException: Provided object has unexpected type!
	kUsageViolation= kRuntimeFault + 2 * ErrorSubClass	UsageViolationException: Violation of allowed usage for the object!
	kAccessViolation= kRuntimeFault + 3 * ErrorSubClass	AccessViolationException: Access rights violation!

▽

△

Header file:	#include "ara/crypto/common/exceptions.h"
Description:	Enumeration of all Security Error Code values that may be reported by ara::crypto.
Notes:	Storage type: 32 bit signed integer (ara::core::ErrorDomain::CodeType).

⌋*(RS_CRYPTO_02310)*

8.10.3 Function Documentation

8.10.3.1 constexpr ara::core::ErrorDomain const& ara::crypto::GetSecurityErrorDomain ( )

[SWS_CRYPT_19950]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::GetSecurityErrorDomain()	
Scope:	namespace ara::crypto	
Syntax:	`inline constexpr ara::core::ErrorDomain const& GetSecurityErrorDomain ();`	
Return value:	ara::core::ErrorDomain const &	constant reference to the single instance of the SecurityErrorDomain
Header file:	#include "ara/crypto/common/exceptions.h"	
Description:	Singleton factory function of the SecurityErrorDomain.	

⌋*(RS_CRYPTO_02310)*

8.10.3.2 constexpr ara::core::ErrorCode ara::crypto::MakeErrorCode (SecurityErrorDomain::Errc *code*, ara::core::ErrorDomain::SupportDataType *data*, char const ∗ *message* = `nullptr`)

[SWS_CRYPT_19951]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::MakeErrorCode(SecurityErrorDomain::Errc code, ara::core::ErrorDomain::SupportDataType data, char const *message=nullptr)	
Scope:	namespace ara::crypto	
Syntax:	`inline constexpr ara::core::ErrorCode MakeErrorCode (SecurityErrorDomain::Errc code, ara::core::ErrorDomain::SupportDataType data, char const *message=nullptr);`	
Parameters (in):	code	an error code identifier from the SecurityErrc enumeration
	data	supplementary data for the error description

▽

△

	message	additional error message supplied by user-code
Return value:	ara::core::ErrorCode	an instance of ErrorCode created according the arguments
Header file:	#include "ara/crypto/common/exceptions.h"	
Description:	Makes Error Code instances from the Security Error Domain.	
Notes:	The returned ErrorCode instance always references to SecurityErrorDomain.	

⌋*(RS_CRYPTO_02310)*

8.11 Crypto Provider API

Modules

- Elementary types
- Basic cryptographic interfaces
- Top-level cryptographic interfaces

Namespaces

- ara::crypto::cryp

Functions

- ara::core::Result< CryptoProvider::Sptr > ara::crypto::cryp::LoadCryptoProvider (const CryptoProviderUid ∗providerUid=nullptr) noexcept

Detailed Description

Crypto Provider is a component of Crypto Stack that implements cryptographic primitives (algorithms) supported by the stack. Implementation of this component may be software or hardware based (HSM/TPM), but in both cases design of this component must isolate consumer applications from direct access to confidential key material (secret or private keys).

Note

Crypto Stack may incorporate a few Crypto Providers representing different software/hardware implementations.

8.11.1 Function Documentation

8.11.1.1 ara::core::Result<CryptoProvider::Sptr> ara::crypto::cryp::LoadCryptoProvider (const CryptoProviderUid ∗ *providerUid* = `nullptr`) `[noexcept]`

[SWS_CRYPT_20099]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::LoadCryptoProvider(const CryptoProviderUid *providerUid=nullptr)	
Scope:	namespace ara::crypto::cryp	
Syntax:	`ara::core::Result<CryptoProvider::Sptr> LoadCryptoProvider (const CryptoProviderUid *providerUid=nullptr) noexcept;`	
Parameters (in):	providerUid	the globally unique identifier of required Crypto Provider
Return value:	ara::core::Result< CryptoProvider::Sptr >	shared smart pointer to loaded Crypto Provider
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/entry_point.h"	
Description:	Factory that creates or return existing single instance of specific Crypto Provider.	
Notes:	If (providerUid == nullptr) then platform default provider should be loaded. [Error]: SecurityErrorDomain::kRuntimeFault if requested Crypto Provider cannot be loaded	

⌋(*RS_CRYPTO_02401*)

8.12 Elementary types

Typedefs

- using ara::crypto::cryp::ReservedContextIndex = std::size_t
- using ara::crypto::cryp::ReservedObjectIndex = std::size_t

Enumerations

Variables

- static const ReservedContextIndex ara::crypto::cryp::kAllocContextOnHeap = static_cast<ReservedContextIndex>(-1)
- static const ReservedObjectIndex ara::crypto::cryp::kAllocObjectOnHeap = static_cast<ReservedObjectIndex>(-1)

Detailed Description

This group includes elementary types definitions specific for Crypto Provider.

8.12.1 Typedef Documentation

8.12.1.1 using ara::crypto::cryp::ReservedContextIndex = typedef std::size_t

[SWS_CRYPT_20002]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::ReservedContextIndex
Scope:	namespace ara::crypto::cryp
Derived from:	typedef std::size_t
Syntax:	`using ara::crypto::cryp::ReservedContextIndex = std::size_t;`
Header file:	#include "ara/crypto/cryp/memory_pool.h"
Description:	Type of Reserved Context Index (maximal value means "NOT RESERVED").
Notes:	Values of this type should be used for indexing slots reserved for Cryptographic Contexts in the "Memory Pool".

⌋*(RS_CRYPTO_02311)*

8.12.1.2 using ara::crypto::cryp::ReservedObjectIndex = typedef std::size_t

[SWS_CRYPT_20003]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::ReservedObjectIndex
Scope:	namespace ara::crypto::cryp
Derived from:	typedef std::size_t
Syntax:	`using ara::crypto::cryp::ReservedObjectIndex = std::size_t;`
Header file:	#include "ara/crypto/cryp/memory_pool.h"
Description:	Type of Reserved Object Index (maximal value means "NOT RESERVED").
Notes:	Values of this type should be used for indexing slots reserved for Cryptographic Objects in the "Memory Pool".

⌋*(RS_CRYPTO_02311)*

8.12.2 Enumeration Type Documentation

8.12.2.1 enum ara::crypto::cryp::KeyType : std::uint8_t `[strong]`

[SWS_CRYPT_20001]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::cryp::KeyType	
Scope:	namespace ara::crypto::cryp	
Values:	kUnknown= 0	A value reserved for erroneous situations.
	kSymmetricKey= 1	Symmetric key (SymmetricKey interface).
	kPrivateKey= 2	Private key (PrivateKey interface).
	kPublicKey= 3	Public key (PublicKey interface).
Header file:	#include "ara/crypto/cryp/key_type.h"	
Description:	Enumeration of all known types of supported key objects.	
Notes:	Storage type: 8 bit unsigned integer.	

⌋*(RS_CRYPTO_02311)*

8.12.3 Variable Documentation

8.12.3.1 const ReservedContextIndex ara::crypto::cryp::kAllocContextOnHeap = static_cast<ReservedContextIndex>(-1) `[static]`

[SWS_CRYPT_20004]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::cryp::kAllocContextOnHeap
Scope:	namespace ara::crypto::cryp
Type:	const ReservedContextIndex
Syntax:	`const ReservedContextIndex ara::crypto::cryp::kAllocContextOnHeap= static_cast<ReservedContextIndex>(-1);`
Header file:	#include "ara/crypto/cryp/memory_pool.h"
Description:	Reserved index value in order to inform provider that requested Context should be allocated on the heap.

⌋(*RS_CRYPTO_02311*)

8.12.3.2 const ReservedObjectIndex ara::crypto::cryp::kAllocObjectOnHeap = static_cast<ReservedObjectIndex>(-1) `[static]`

[SWS_CRYPT_20005]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::cryp::kAllocObjectOnHeap
Scope:	namespace ara::crypto::cryp
Type:	const ReservedObjectIndex
Syntax:	`const ReservedObjectIndex ara::crypto::cryp::kAllocObjectOnHeap= static_cast<ReservedObjectIndex>(-1);`
Header file:	#include "ara/crypto/cryp/memory_pool.h"
Description:	Reserved index value in order to inform provider that requested Object should be allocated on the heap.

⌋(*RS_CRYPTO_02311*)

8.13 Basic cryptographic interfaces

Modules

- Basic object interfaces
- Basic transformation interfaces

Classes

- class ara::crypto::cryp::CryptoPrimitiveId

Detailed Description

This group includes only basic cryptographic interfaces inherited and shared by top-level interfaces of the Crypto Provider API.

8.13.1 Class Documentation

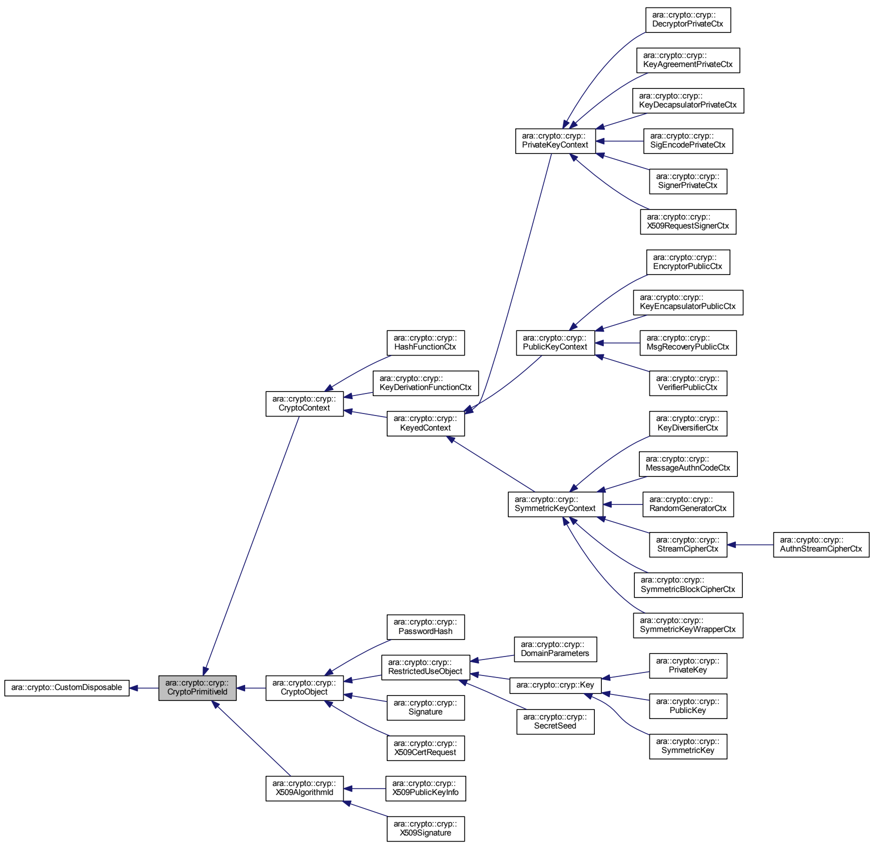
8.13.1.1 class ara::crypto::cryp::CryptoPrimitiveId

[SWS_CRYPT_20600]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02311)*

Inheritance diagram for ara::crypto::cryp::CryptoPrimitiveId:


ara::crypto::cryp::DecryptorPrivateCtx
ara::crypto::cryp::KeyAgreementPrivateCtx
ara::crypto::cryp::KeyDecapsulatorPrivateCtx
ara::crypto::cryp::PrivateKeyContext
ara::crypto::cryp::SigEncodePrivateCtx
ara::crypto::cryp::SignerPrivateCtx
ara::crypto::cryp::X509RequestSignerCtx
ara::crypto::cryp::EncryptorPublicCtx
ara::crypto::cryp::KeyEncapsulatorPublicCtx
ara::crypto::cryp::HashFunctionCtx
ara::crypto::cryp::PublicKeyContext
ara::crypto::cryp::MsgRecoveryPublicCtx
ara::crypto::cryp::KeyDerivationFunctionCtx
ara::crypto::cryp::VerifierPublicCtx
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::KeyedContext
ara::crypto::cryp::KeyDiversifierCtx
ara::crypto::cryp::MessageAuthnCodeCtx
ara::crypto::cryp::SymmetricKeyContext
ara::crypto::cryp::RandomGeneratorCtx
ara::crypto::cryp::StreamCipherCtx
ara::crypto::cryp::AuthnStreamCipherCtx
ara::crypto::cryp::SymmetricBlockCipherCtx
ara::crypto::cryp::SymmetricKeyWrapperCtx
ara::crypto::cryp::PasswordHash
ara::crypto::cryp::DomainParameters
ara::crypto::cryp::RestrictedUseObject
ara::crypto::cryp::PrivateKey
ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoObject
ara::crypto::cryp::Key
ara::crypto::cryp::Signature
ara::crypto::cryp::SecretSeed
ara::crypto::cryp::PublicKey
ara::crypto::cryp::SymmetricKey
ara::crypto::cryp::X509CertRequest
ara::crypto::cryp::X509AlgorithmId
ara::crypto::cryp::X509PublicKeyInfo
ara::crypto::cryp::X509Signature

Public Types

- using AlgId = CryptoAlgId

Public Member Functions

- virtual const ara::core::StringView GetPrimitiveName () const noexcept=0
- virtual AlgId GetPrimitiveId () const noexcept=0
- virtual Category GetCategory () const noexcept=0
- virtual CryptoProvider & MyProvider () const noexcept=0

Additional Inherited Members

8.13.1.1.1 Member Typedef Documentation

8.13.1.1.1.1 using ara::crypto::cryp::CryptoPrimitiveId::AlgId = CryptoAlgId

[SWS_CRYPT_20601]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoPrimitiveId::AlgId
Scope:	class ara::crypto::cryp::CryptoPrimitiveId
Derived from:	CryptoAlgId
Syntax:	`using ara::crypto::cryp::CryptoPrimitiveId::AlgId = CryptoAlgId;`
Header file:	#include "ara/crypto/cryp/crypto_primitive_id.h"
Description:	Type definition of vendor specific binary Crypto Primitive ID.

⌋*(RS_CRYPTO_02311)*

8.13.1.1.2 Member Enumeration Documentation

8.13.1.1.2.1 enum ara::crypto::cryp::CryptoPrimitiveId::Category : std::uint8_t `[strong]`

[SWS_CRYPT_20602]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::cryp::CryptoPrimitiveId::Category	
Scope:	class ara::crypto::cryp::CryptoPrimitiveId	
Values:	kUnknown= 0	A value reserved for erroneous situations.
	kGenericSymmetricKey= 1	Generic set of symmetric key primitives (Symmetric Key interface). This category is applicable to key objects only!
	kGenericAsymmetricDlp= 2	Generic set of public / private key primitives based on the Disrete Logarifm Problem (DLP), i.e. interfaces: PublicKey / PrivateKey. This category is applicable to key objects only!
	kGenericAsymmetricIfp= 3	Generic set of public / private key primitives based on the Integer Factoring Problem (IFP), i.e. interfaces: PublicKey / PrivateKey. This category is applicable to key objects only!
	kHashFunction= 4	Keyless hash function primitives (HashFunctionCtx interface).
	kKeyDerivationFunction= 5	Keyless key derivation function (KDF) primitives (KeyDeriveFuncCtx interface).
	kSymmetricBlockCipher= 6	Symmetric symmetric block cipher primitives (SymmetricBlockCipherCtx interface).

▽

△

	kSymmetricStreamCipher= 7	Symmetric stream cipher primitives (StreamCipher Ctx interface).
	kSymmetricAuthentication= 8	Symmetric message authetication code (MAC) primitives (MessageAuthnCodeCtx interface).
	kAuthenticStreamCipher= 9	Symmetric authenticated stream cipher primitives (AuthnStreamCipherCtx interface).
	kKeyDiversification= 10	Symmetric key diversifier primitives (KeyDiversifier Ctx interface).
	kSymmetricKeyWrap= 11	Symmetric symmetric key wrapping primitives (SymmetricKeyWrapCtx interface).
	kRandomGenerator= 12	Random number generator (RNG) primitives (RandomGeneratorCtx interface).
	kKeyAgreementDlp= 13	Asymmetric key agreement primitives, based on the DLP (KeyAgreePrivateCtx interface).
	kDigitalSignatureDlp= 14	Asymmetric signature primitives, based on the DLP. Signature calculation and verification interfaces: SignPrivateCtx / VerifyPublicCtx.
	kSignatureEncoderIfp= 15	Asymmetric signature encoding primitives with message recovery, based on the IFP. Signature calculation and message recovery interfaces: Sig EncodePrivateCtx / MsgRecoveryPublicCtx.
	kAsymmetricCipherIfp= 16	Asymmetric cipher primitives, based on the IFP. Encryption / decryption interfaces: EncryptPublicCtx / DecryptPrivateCtx.
	kKeyEncapsulationIfp= 17	Asymmetric key encapsulation primitives, based on the IFP. Encapsulation / Decapsulation interfaces: KeyEncapsulatePublicCtx / KeyDecapsulatePrivate Ctx.
Header file:	#include "ara/crypto/cryp/crypto_primitive_id.h"	
Description:	Enumeration of categories of all supported crypto primitives.	
Notes:	Storage type: 8 bit unsigned integer.	

⌋*(RS_CRYPTO_02311)*

8.13.1.1.3 Member Function Documentation

8.13.1.1.3.1 virtual const ara::core::StringView ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveName ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_20611]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoPrimitiveId::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/crypto_primitive_id.h"	
Description:	Get a unified name of the primitive.	
Notes:	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 Crypto PrimitiveId instance!	

⌋(*RS_CRYPTO_02308*)

8.13.1.1.3.2 virtual AlgId ara::crypto::cryp::CryptoPrimitiveId::GetPrimitiveId ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20612]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoPrimitiveId::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/crypto_primitive_id.h"	
Description:	Get vendor specific ID of the primitive.	

⌋(*RS_CRYPTO_02309*)

8.13.1.1.3.3 virtual Category ara::crypto::cryp::CryptoPrimitiveId::GetCategory ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20613]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoPrimitiveId::GetCategory()	
Scope:	class ara::crypto::cryp::CryptoPrimitiveId	
Syntax:	`virtual Category GetCategory () const noexcept=0;`	
Return value:	Category	the category of the primitive
Exception Safety:	noexcept	

▽

△

Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/crypto_primitive_id.h"
Description:	Get the category of the primitive.

⌋*(RS_CRYPTO_02309)*

8.13.1.1.3.4 virtual CryptoProvider& ara::crypto::cryp::CryptoPrimitiveId::MyProvider ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20614]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02401)*

8.14 Basic object interfaces

Classes

- class ara::crypto::cryp::CryptoObject
- class ara::crypto::cryp::Key
- class ara::crypto::cryp::KeyMaterial
- class ara::crypto::cryp::RestrictedUseObject
- class ara::crypto::cryp::X509AlgorithmId

Detailed Description

This group includes basic interfaces of cryptographic objects that build a basis for top-level interfaces of cryptographic objects. Cryptographic object is a "passive" cryptographic artifact like: key, seed, domain parameters, signature, etc. A list of all crypto object types supported by Crypto Stack is presented by the enumeration `CryptoObjectType`.

8.14.1 Class Documentation

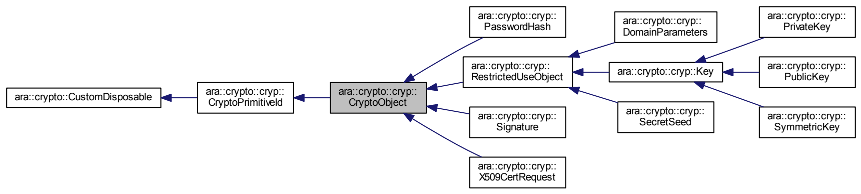
8.14.1.1 class ara::crypto::cryp::CryptoObject

[SWS_CRYPT_20500]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02311)*

Inheritance diagram for ara::crypto::cryp::CryptoObject:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoObject
ara::crypto::cryp::PasswordHash
ara::crypto::cryp::RestrictedUseObject
ara::crypto::cryp::Signature
ara::crypto::cryp::X509CertRequest
ara::crypto::cryp::DomainParameters
ara::crypto::cryp::Key
ara::crypto::cryp::SecretSeed
ara::crypto::cryp::PrivateKey
ara::crypto::cryp::PublicKey
ara::crypto::cryp::SymmetricKey

Public Types

- using Uptr = std::unique_ptr< CryptoObject, CustomDeleter >
- using Uptrc = std::unique_ptr< const CryptoObject, CustomDeleter >
- using Type = CryptoObjectType

Public Member Functions

- virtual Type GetObjectType () const noexcept=0
- virtual bool IsSession () const noexcept=0
- virtual bool IsExportable () const noexcept=0
- virtual bool GetObjectId (CryptoObjectUid ∗objectId=nullptr) const noexcept=0
- virtual Type HasDependence (CryptoObjectUid ∗objectId=nullptr) const noexcept=0
- virtual std::size_t StorageSize () const noexcept=0
- virtual ara::core::Result< void > Save (TrustedContainer &container, TrustedContainer ∗referenced=nullptr) const noexcept=0

Static Public Member Functions

- template<class ConcreteObject >
  static ara::core::Result< typename ConcreteObject::Uptrc > Downcast (CryptoObject::Uptrc &&object) noexcept

Additional Inherited Members

8.14.1.1.1 Member Typedef Documentation

8.14.1.1.1.1 using ara::crypto::cryp::CryptoObject::Uptr = std::unique_ptr<CryptoObject, CustomDeleter>

[SWS_CRYPT_20501]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoObject::Uptr
Scope:	class ara::crypto::cryp::CryptoObject
Derived from:	std::unique_ptr<CryptoObject, CustomDeleter>
Syntax:	`using ara::crypto::cryp::CryptoObject::Uptr = std::unique_ptr<CryptoObject, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/crypto_object.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.14.1.1.1.2 using ara::crypto::cryp::CryptoObject::Uptrc = std::unique_ptr<const CryptoObject, CustomDeleter>

[SWS_CRYPT_20502]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoObject::Uptrc
Scope:	class ara::crypto::cryp::CryptoObject
Derived from:	std::unique_ptr<const CryptoObject, CustomDeleter>
Syntax:	`using ara::crypto::cryp::CryptoObject::Uptrc = std::unique_ptr<const CryptoObject, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/crypto_object.h"
Description:	Unique smart pointer of the constant interface.

⌋*(RS_CRYPTO_02404)*

8.14.1.1.1.3 using ara::crypto::cryp::CryptoObject::Type = CryptoObjectType

[SWS_CRYPT_20503]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoObject::Type
Scope:	class ara::crypto::cryp::CryptoObject
Derived from:	CryptoObjectType
Syntax:	`using ara::crypto::cryp::CryptoObject::Type = CryptoObjectType;`
Header file:	#include "ara/crypto/cryp/crypto_object.h"
Description:	Enumeration of all types of crypto objects.

⌋*(RS_CRYPTO_02311)*

8.14.1.1.2 Member Function Documentation

8.14.1.1.2.1 virtual Type ara::crypto::cryp::CryptoObject::GetObjectType ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_20511]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoObject::GetObjectType()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual Type GetObjectType () const noexcept=0;`	
Return value:	Type	one of object types except Type::kUnknown
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_object.h"	
Description:	Return the type of this object.	

⌋*(RS_CRYPTO_02311)*

8.14.1.1.2.2 virtual bool ara::crypto::cryp::CryptoObject::IsSession ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_20512]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoObject::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/crypto_object.h"
Description:	Return the "session" (or "temporary") attribute of the object.
Notes:	A temporary object cannot be saved to a non-volatile trusted container! 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)*

8.14.1.1.2.3 virtual bool ara::crypto::cryp::CryptoObject::IsExportable ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_20513]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoObject::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/crypto_object.h"	
Description:	Get the exportability attribute of the crypto object.	
Notes:	An exportable object must have an assigned COUID (see GetObjectId()).	

⌋*(RS_CRYPTO_02113)*

8.14.1.1.2.4 virtual bool ara::crypto::cryp::CryptoObject::GetObjectId (CryptoObjectUid ∗ *objectId = nullptr*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_20514]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoObject::GetObjectId(CryptoObjectUid *objectId=nullptr)
Scope:	class ara::crypto::cryp::CryptoObject

▽

△

Syntax:	`virtual bool GetObjectId (CryptoObjectUid *objectId=nullptr) const noexcept=0;`	
Parameters (out):	objectId	optional pointer to a buffer for the object's UID saving
Return value:	bool	true if the object has assigned COUID and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_object.h"	
Description:	Return the object's UID if it is assigned to the object.	
Notes:	An object that has no an assigned COUID cannot be (securely) serialized / exported or saved to a non-volatile container. An object should not have a COUID if it is session and non-exportable simultaneously or if it is incomplete yet (last is applicable for domain parameters only). 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)*

8.14.1.1.2.5 virtual Type ara::crypto::cryp::CryptoObject::HasDependence (CryptoObjectUid ∗ *objectId* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20515]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoObject::HasDependence(CryptoObjectUid *objectId=nullptr)	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual Type HasDependence (CryptoObjectUid *objectId=nullptr) const noexcept=0;`	
Parameters (out):	objectId	the optional pointer to a buffer for the target Crypto Object UID (COUID)
Return value:	Type	target object type for existing dependence or Type::kUnknown if the current object doesn't depend from any other one
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_object.h"	
Description:	Return an identifier of an object from which depends the current one.	
Notes:	If (objectId != nullptr) but the current object has no dependence from other one then the objectId should be filled by zeros. For signatures objects this method must return a reference to correspondent signature verification public key! For keys objects this method should return a reference to domain parameters. ▽	

▽

△

	△ For domain parameters object this method (optionally) can return a reference to another domain parameters required for this one. Unambiguous identification of a crypto object requires both components: CryptoObjectUid and Type. A caller code may omit the optional argument if only the dependency type is interested.

⌋(*RS_CRYPTO_02005*)

8.14.1.1.2.6 virtual std::size_t ara::crypto::cryp::CryptoObject::StorageSize ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20516]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoObject::StorageSize()	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual std::size_t StorageSize () const noexcept=0;`	
Return value:	std::size_t	size in bytes of the objects required for its storage
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_object.h"	
Description:	Return a storage size of the object.	

⌋(*RS_CRYPTO_02309*)

8.14.1.1.2.7 virtual ara::core::Result<void> ara::crypto::cryp::CryptoObject::Save (TrustedContainer & *container*, TrustedContainer ∗ *referenced* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20517]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoObject::Save(TrustedContainer &container, TrustedContainer *referenced=nullptr)	
Scope:	class ara::crypto::cryp::CryptoObject	
Syntax:	`virtual ara::core::Result<void> Save (TrustedContainer &container, TrustedContainer *referenced=nullptr) const noexcept=0;`	
Parameters (in):	container	the target trusted container
	referenced	a pointer to another trusted container that keeps referenced crypto object

▽

△

Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_object.h"	
Description:	Save itself to provided trusted container.	
Notes:	Both specified containers (target and referenced) must have non-read-only status! Actually it means that the both containers must be "owned" by the current Actor (caller application/process). The ownership status may be verified by the method see Trusted Container::HasOwnership(). Only a non-session/non-temporary and completed object (i.e. that have a COUID) can be saved! If (referenced != nullptr) then internal "references counter" of the referenced container must be incremented! If the target container argument represents a persistent key slot then the referenced argument must also represent an opened "for update" persistent key slot (see KeyStorageProvider::Open AsOwner())! I.e. saving an object to persistent key storage with reference to volatile container is prohibited! If the target container is volatile then the referenced argument may represent as a volatile, so a persistent container. If (referenced != nullptr) and the target container argument represents a volatile container, then the "references counter" of the referenced container will be decremented during destroy or update of an object saved to the target container. Content of a referenced container cannot be changed while its "references counter" is non-zero! Also see keys::KeyStorageProvider::FindObject() for details of an object search in the Key Storage. [Error]: SecurityErrorDomain::kAccessViolation if the target or the referenced non-volatile containers were opened "for usage", i.e. if (!container.HasOwnership() || !referenced->Has Ownership()) [Error]: SecurityErrorDomain::kIncompatibleObject if the object is "session", but the container is non-volatile [Error]: SecurityErrorDomain::kContentRestrictions if the object doesn't satisfy the slot restrictions ([Error]: SecurityErrorDomain::kInsufficientCapacity if the capacity of the target container is not enough, i.e. if (container.Capacity() < this->StorageSize()) [Error]: SecurityErrorDomain::kBadObjectReference if (referenced != nullptr), but this crypto object doesn't support referencing to other objects;if (referenced != nullptr), but an object in referenced container cannot be referenced;if (referenced != nullptr), but this crypto object and referenced one have incompatible for referencing types and/or algorithms;if (referenced != nullptr) and referenced is volatile, but container is non-volatile [Error]: SecurityErrorDomain::kEmptyContainer if (referenced != nullptr), but the referenced trusted container is empty	

⌋*(RS_CRYPTO_02004)*

8.14.1.1.2.8 template<class ConcreteObject > static ara::core::Result<typename ConcreteObject::Uptrc> ara::crypto::cryp::CryptoObject::Downcast (CryptoObject::Uptrc && *object*) [static], [noexcept]

[SWS_CRYPT_20518]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoObject::Downcast(CryptoObject::Uptrc &&object)	
Scope:	class ara::crypto::cryp::CryptoObject	

Syntax:	`template <class ConcreteObject>`
	`inline 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	
Header file:	#include "ara/crypto/cryp/crypto_object.h"	
Description:	Downcast and move unique smart pointer from the generic CryptoObject interface to concrete derived object.	
Notes:	[Error]: SecurityErrorDomain::kBadObjectType if an actual type of the object is not the specified ConcreteObject	

⌋*(RS_CRYPTO_02311)*

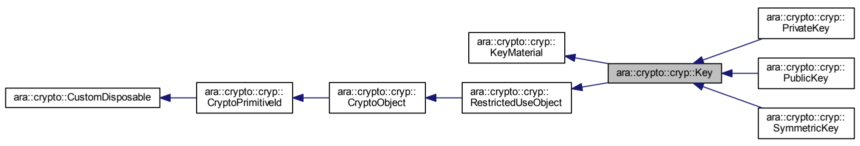
8.14.1.2 class ara::crypto::cryp::Key

[SWS_CRYPT_21200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::Key
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::KeyMaterial
Syntax:	`class Key : public KeyMaterial {...};`
Header file:	#include "ara/crypto/cryp/key.h"
Description:	Generalized Key object interface.
Notes:	An implementation can filter allowed key values at generation/derivation time in order to prevent production of "weak" and "semi-weak" keys patterns specific to concrete algorithm.

⌋*(RS_CRYPTO_02403)*

Inheritance diagram for ara::crypto::cryp::Key:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoObject
ara::crypto::cryp:: RestrictedUseObject
ara::crypto::cryp:: KeyMaterial
ara::crypto::cryp::Key
ara::crypto::cryp:: PrivateKey
ara::crypto::cryp:: PublicKey
ara::crypto::cryp:: SymmetricKey

Public Types

- using Uptrc = std::unique_ptr< const Key, CustomDeleter >

Public Member Functions

- virtual bool IsCompatible (AlgId algId) const noexcept=0
- virtual bool IsCompatible (const KeyedContext &context) const noexcept=0
- virtual KeyType GetKeyType () const noexcept=0
- virtual bool IsPublic () const noexcept=0

Additional Inherited Members

8.14.1.2.1 Member Typedef Documentation

8.14.1.2.1.1 using ara::crypto::cryp::Key::Uptrc = std::unique_ptr<const Key, CustomDeleter>

[SWS_CRYPT_21201]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::Key::Uptrc
Scope:	class ara::crypto::cryp::Key
Derived from:	std::unique_ptr<const Key, CustomDeleter>
Syntax:	`using ara::crypto::cryp::Key::Uptrc = std::unique_ptr<const Key, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/key.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.14.1.2.2 Member Function Documentation

8.14.1.2.2.1 virtual bool ara::crypto::cryp::Key::IsCompatible (AlgId *algId*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21212]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::Key::IsCompatible(AlgId algId)	
Scope:	class ara::crypto::cryp::Key	
Syntax:	`virtual bool IsCompatible (AlgId algId) const noexcept=0;`	
Parameters (in):	algId	target Algorithm ID
Return value:	bool	true if the key is compatible with the algorithm specified by the algId and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key.h"	
Description:	Check compatibility of this key with an algorithm specified by an ID.	

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02309)*

8.14.1.2.2.2 virtual bool ara::crypto::cryp::Key::IsCompatible (const KeyedContext & *context*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_21213]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::Key::IsCompatible(const KeyedContext &context)	
Scope:	class ara::crypto::cryp::Key	
Syntax:	`virtual bool IsCompatible (const KeyedContext &context) const noexcept=0;`	
Parameters (in):	context	target keyed context
Return value:	bool	true if the key is compatible with the transformation of the context and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key.h"	
Description:	Check compatibility of this key with a crypto transformation configured in the keyed context.	
Notes:	This method compares not only the Crypto Primitive IDs, but also the COUID of domain parameters objects associated with the key object and with provided context! If the COUIDs differ then this method returns false.	

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02309)*

8.14.1.2.2.3 virtual KeyType ara::crypto::cryp::Key::GetKeyType ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_21214]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::Key::GetKeyType()	
Scope:	class ara::crypto::cryp::Key	
Syntax:	`virtual KeyType GetKeyType () const noexcept=0;`	
Return value:	KeyType	identifier of the top-level interface type of the key
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key.h"	
Description:	Get actual type of this key object.	

⌋*(RS_CRYPTO_02309)*

8.14.1.2.2.4 virtual bool ara::crypto::cryp::Key::IsPublic ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_21215]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::Key::IsPublic()	
Scope:	class ara::crypto::cryp::Key	
Syntax:	`virtual bool IsPublic () const noexcept=0;`	
Return value:	bool	true if the key is public and false if the key is private or secret
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key.h"	
Description:	Get publicity of the key.	

⌋*(RS_CRYPTO_02309)*

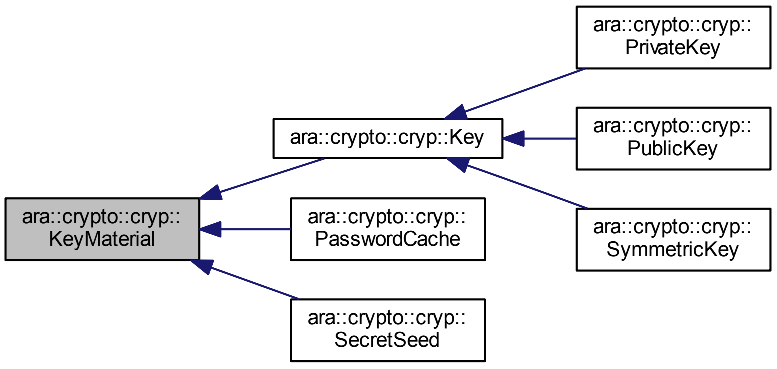
8.14.1.3 class ara::crypto::cryp::KeyMaterial

[SWS_CRYPT_22000]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyMaterial
Scope:	namespace ara::crypto::cryp
Syntax:	`class KeyMaterial {...};`
Header file:	#include "ara/crypto/cryp/key_material.h"
Description:	Generalized Key Material interface (in general case it represents a non-saveable temporary entity).

⌋*(RS_CRYPTO_02311)*

Inheritance diagram for ara::crypto::cryp::KeyMaterial:


ara::crypto::cryp::PrivateKey
ara::crypto::cryp::Key
ara::crypto::cryp::PublicKey
ara::crypto::cryp::KeyMaterial
ara::crypto::cryp::PasswordCache
ara::crypto::cryp::SymmetricKey
ara::crypto::cryp::SecretSeed

Public Member Functions

- virtual std::size_t GetActualKeyBitLength () const noexcept=0

Protected Member Functions

- virtual ~KeyMaterial ()=default

8.14.1.3.1 Constructor & Destructor Documentation

8.14.1.3.1.1 virtual ara::crypto::cryp::KeyMaterial::~KeyMaterial ( ) `[protected], [virtual], [default]`

[SWS_CRYPT_22010]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::KeyMaterial::~KeyMaterial()
Scope:	class ara::crypto::cryp::KeyMaterial
Visibility:	protected
Syntax:	`virtual ~KeyMaterial ()=default;`
Header file:	#include "ara/crypto/cryp/key_material.h"
Description:	Virtual destructor.

⌋*(RS_CRYPTO_02311)*

8.14.1.3.2 Member Function Documentation

8.14.1.3.2.1 virtual std::size_t ara::crypto::cryp::KeyMaterial::GetActualKeyBitLength ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22011]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyMaterial::GetActualKeyBitLength()	
Scope:	class ara::crypto::cryp::KeyMaterial	
Syntax:	`virtual std::size_t GetActualKeyBitLength () const noexcept=0;`	
Return value:	std::size_t	actual key length in bits
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_material.h"	
Description:	Get actual key length in bits.	

⌋*(RS_CRYPTO_02309)*

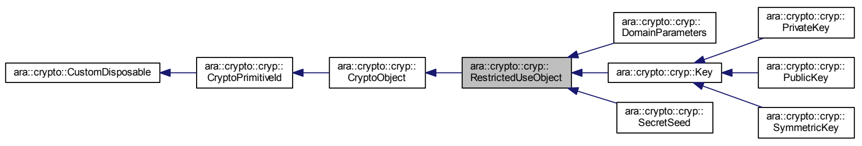
8.14.1.4 class ara::crypto::cryp::RestrictedUseObject

[SWS_CRYPT_24800]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02008)*

Inheritance diagram for ara::crypto::cryp::RestrictedUseObject:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoObject
ara::crypto::cryp::RestrictedUseObject
ara::crypto::cryp::DomainParameters
ara::crypto::cryp::Key
ara::crypto::cryp::SecretSeed
ara::crypto::cryp::PrivateKey
ara::crypto::cryp::PublicKey
ara::crypto::cryp::SymmetricKey

Public Types

- using Usage = AllowedUsageFlags

Public Member Functions

- virtual Usage GetAllowedUsage () const noexcept=0

Additional Inherited Members

8.14.1.4.1 Member Typedef Documentation

8.14.1.4.1.1 using ara::crypto::cryp::RestrictedUseObject::Usage = AllowedUsageFlags

[SWS_CRYPT_24801]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::RestrictedUseObject::Usage
Scope:	class ara::crypto::cryp::RestrictedUseObject
Derived from:	AllowedUsageFlags
Syntax:	`using ara::crypto::cryp::RestrictedUseObject::Usage = AllowedUsageFlags;`
Header file:	#include "ara/crypto/cryp/restricted_use_object.h"
Description:	Alias to the container type for bit-flags of allowed usages of the object.

⌋*(RS_CRYPTO_02008)*

8.14.1.4.2 Member Function Documentation

8.14.1.4.2.1 virtual Usage ara::crypto::cryp::RestrictedUseObject::GetAllowedUsage ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_24811]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::RestrictedUseObject::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/restricted_use_object.h"	
Description:	Get allowed usages of this object.	

▽

△

Notes:	If this method is called for a Domain Parameters object that is not completed yet then it returns kAllowPrototypedOnly.

⌋*(RS_CRYPTO_02008)*

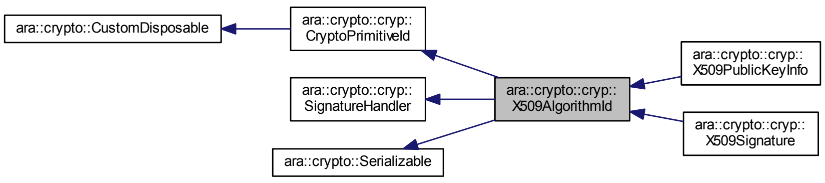
8.14.1.5 class ara::crypto::cryp::X509AlgorithmId

[SWS_CRYPT_24200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::X509AlgorithmId
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::CryptoPrimitiveId
Syntax:	`class X509AlgorithmId : public CryptoPrimitiveId {...};`
Header file:	#include "ara/crypto/cryp/x509_algorithm_id.h"
Description:	X.509 Algorithm ID interface.
Notes:	If instance of this interface is created for unsupported algorithm, then method GetPrimitiveId() will return kAlgIdUnknown!

⌋*(RS_CRYPTO_02307)*

Inheritance diagram for ara::crypto::cryp::X509AlgorithmId:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: SignatureHandler
ara::crypto::Serializable
ara::crypto::cryp:: X509AlgorithmId
ara::crypto::cryp:: X509PublicKeyInfo
ara::crypto::cryp:: X509Signature

Public Types

- using Uptrc = std::unique_ptr< const X509AlgorithmId, CustomDeleter >

Public Member Functions

- virtual bool HasDomainParameters () const noexcept=0
- virtual ara::core::Result< DomainParameters::Sptrc > GetDomainParameters (ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

- virtual bool IsSameParameters (const DomainParameters &params) const noexcept=0

Additional Inherited Members

8.14.1.5.1 Member Typedef Documentation

8.14.1.5.1.1 using ara::crypto::cryp::X509AlgorithmId::Uptrc = std::unique_ptr<const X509AlgorithmId, CustomDeleter>

[SWS_CRYPT_24201]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::X509AlgorithmId::Uptrc
Scope:	class ara::crypto::cryp::X509AlgorithmId
Derived from:	std::unique_ptr<const X509AlgorithmId, CustomDeleter>
Syntax:	`using ara::crypto::cryp::X509AlgorithmId::Uptrc = std::unique_ptr<const X509AlgorithmId, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/x509_algorithm_id.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.14.1.5.2 Member Function Documentation

8.14.1.5.2.1 virtual bool ara::crypto::cryp::X509AlgorithmId::HasDomainParameters ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_24211]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509AlgorithmId::HasDomainParameters()	
Scope:	class ara::crypto::cryp::X509AlgorithmId	
Syntax:	`virtual bool HasDomainParameters () const noexcept=0;`	
Return value:	bool	true if this instance includes domain parameters and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_algorithm_id.h"	
Description:	Verify presence of domain parameters in this object.	

⌋*(RS_CRYPTO_02108)*

8.14.1.5.2.2 virtual ara::core::Result<DomainParameters::Sptrc> ara::crypto::cryp::X509AlgorithmId::GetDomainParameters (ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const [pure virtual], [noexcept]

[SWS_CRYPT_24212]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509AlgorithmId::GetDomainParameters(ReservedObjectIndex reserved Index=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::X509AlgorithmId	
Syntax:	`virtual ara::core::Result<DomainParameters::Sptrc> GetDomainParameters (ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Domain Parameters::Sptrc >	unique smart pointer to the created Domain parameters object associated with the public key of the subject
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_algorithm_id.h"	
Description:	Get domain parameters object associated with the public key of the subject.	
Notes:	This method returns nullptr if this instance does not include domain parameters [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02108, RS_CRYPTO_02306, RS_CRYPTO_02404)*

8.14.1.5.2.3 virtual bool ara::crypto::cryp::X509AlgorithmId::IsSameParameters (const DomainParameters & *params*) const [pure virtual], [noexcept]

[SWS_CRYPT_24213]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::X509AlgorithmId::IsSameParameters(const DomainParameters &params)
Scope:	class ara::crypto::cryp::X509AlgorithmId
Syntax:	`virtual bool IsSameParameters (const DomainParameters &params) const noexcept=0;`

▽

△

Parameters (in):	params	the domain parameters object for comparison
Return value:	bool	true if values of the stored domain parameters and object provided by the argument are identical
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_algorithm_id.h"	
Description:	Verify the sameness of the provided and internally stored domain parameters.	
Notes:	If the domain parameters specified by the params argument has an incomplete state then this method should return false without actual comparison.	

⌋*(RS_CRYPTO_02108)*

8.15 Basic transformation interfaces

Classes

- class ara::crypto::cryp::BlockCryptor
- class ara::crypto::cryp::BufferedDigest
- class ara::crypto::cryp::CryptoContext
- class ara::crypto::cryp::KeyEncapsulator
- class ara::crypto::cryp::KeyedContext
- class ara::crypto::cryp::PrivateKeyContext
- class ara::crypto::cryp::PublicKeyContext
- class ara::crypto::cryp::SignatureHandler
- class ara::crypto::cryp::StreamStarter
- class ara::crypto::cryp::SymmetricKeyContext

Detailed Description

This group includes basic interfaces of cryptographic contexts that build a basis for top-level interfaces of cryptographic contexts. Each cryptographic context represents an internal data structure dedicated for execution of correspondent cryptographic transformation (algorithm calculation).

8.15.1 Class Documentation

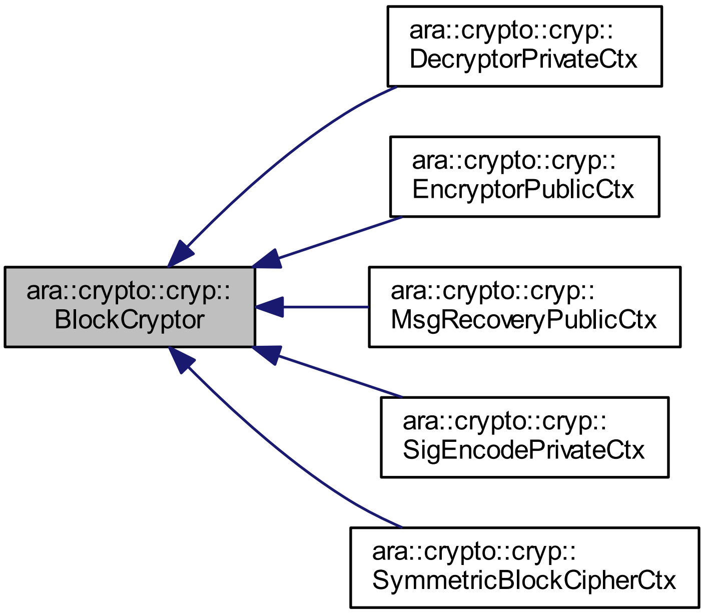
8.15.1.1 class ara::crypto::cryp::BlockCryptor

[SWS_CRYPT_20200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::BlockCryptor
Scope:	namespace ara::crypto::cryp
Syntax:	`class BlockCryptor {...};`
Header file:	#include "ara/crypto/cryp/block_cryptor.h"
Description:	General interface for stateless encryption / decryption of a single data block with padding.
Notes:	The Block Cryptor context should include a defenition of a padding scheme applicable by default. Use non-default value of argument suppressPadding only if you know exactly what you are doing!

⌋*(RS_CRYPTO_02201, RS_CRYPTO_02202)*

Inheritance diagram for ara::crypto::cryp::BlockCryptor:


ara::crypto::cryp:: DecryptorPrivateCtx
ara::crypto::cryp:: EncryptorPublicCtx
ara::crypto::cryp:: BlockCryptor
ara::crypto::cryp:: MsgRecoveryPublicCtx
ara::crypto::cryp:: SigEncodePrivateCtx
ara::crypto::cryp:: SymmetricBlockCipherCtx

Public Member Functions

- virtual bool IsEncryption () const noexcept=0
- bool IsMaxInputOnly () const noexcept
- bool IsMaxOutputOnly () const noexcept
- virtual std::size_t GetMaxInputSize (bool suppressPadding=false) const noexcept=0
- virtual std::size_t GetMaxOutputSize (bool suppressPadding=false) const noexcept=0
- virtual ara::core::Result< std::size_t > ProcessBlock (WritableMemRegion out, ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< void > ProcessBlock (ByteVectorT< Alloc > &out, ReadOnlyMemRegion in, bool suppressPadding=false) const noexcept

Protected Member Functions

- virtual ~BlockCryptor () noexcept=default

8.15.1.1.1 Constructor & Destructor Documentation

8.15.1.1.1.1 virtual ara::crypto::cryp::BlockCryptor::∼BlockCryptor ( ) [protected], [virtual], [default], [noexcept]

[SWS_CRYPT_20210]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::BlockCryptor::~BlockCryptor()
Scope:	class ara::crypto::cryp::BlockCryptor
Visibility:	protected
Syntax:	`virtual ~BlockCryptor () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/crypto/cryp/block_cryptor.h"
Description:	Virtual destructor.

⌋*(RS_CRYPTO_02311)*

8.15.1.1.2 Member Function Documentation

8.15.1.1.2.1 virtual bool ara::crypto::cryp::BlockCryptor::IsEncryption ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_20211]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BlockCryptor::IsEncryption()	
Scope:	class ara::crypto::cryp::BlockCryptor	
Syntax:	`virtual bool IsEncryption () const noexcept=0;`	
Return value:	bool	true if the Block Cryptor context is configured for encryption and false for decryption
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/block_cryptor.h"	
Description:	Return the transformation direction of the current configuration of the Block Cryptor context.	

⌋*(RS_CRYPTO_02309)*

8.15.1.1.2.2 bool ara::crypto::cryp::BlockCryptor::IsMaxInputOnly ( ) const [noexcept]

[SWS_CRYPT_20212]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BlockCryptor::IsMaxInputOnly()	
Scope:	class ara::crypto::cryp::BlockCryptor	
Syntax:	`inline bool IsMaxInputOnly () const noexcept;`	
Return value:	bool	true if the transformation requires the maximum size of input data and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/block_cryptor.h"	
Description:	Indicate that the currently configured transformation accepts only complete blocks of input data.	

⌋*(RS_CRYPTO_02309)*

8.15.1.1.2.3 bool ara::crypto::cryp::BlockCryptor::IsMaxOutputOnly ( ) const [noexcept]

[SWS_CRYPT_20213]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BlockCryptor::IsMaxOutputOnly()	
Scope:	class ara::crypto::cryp::BlockCryptor	
Syntax:	`inline bool IsMaxOutputOnly () const noexcept;`	
Return value:	bool	true if the transformation can produce only the maximum size of output data and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/block_cryptor.h"	
Description:	Indicate that the currently configured transformation can produce only complete blocks of output data.	

⌋*(RS_CRYPTO_02309)*

8.15.1.1.2.4 virtual std::size_t ara::crypto::cryp::BlockCryptor::GetMaxInputSize (bool *suppressPadding* = `false`) const [pure virtual], [noexcept]

[SWS_CRYPT_20214]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::BlockCryptor::GetMaxInputSize(bool suppressPadding=false)
Scope:	class ara::crypto::cryp::BlockCryptor

▽

△

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/block_cryptor.h"	
Description:	Get maximum expected size of the input data block.	
Notes:	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*)

8.15.1.1.2.5 virtual std::size_t ara::crypto::cryp::BlockCryptor::GetMaxOutputSize (bool *suppressPadding* = `false`) const `[pure virtual], [noexcept]`

[SWS_CRYPT_20215]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BlockCryptor::GetMaxOutputSize(bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::BlockCryptor	
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/block_cryptor.h"	
Description:	Get maximum possible size of the output data block.	
Notes:	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*)

8.15.1.1.2.6 virtual ara::core::Result<std::size_t> ara::crypto::cryp::BlockCryptor::ProcessBlock (WritableMemRegion *out*, ReadOnlyMemRegion *in*, bool *suppressPadding* = `false`) const `[pure virtual], [noexcept]`

[SWS_CRYPT_20216]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BlockCryptor::ProcessBlock(WritableMemRegion out, ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::BlockCryptor	
Syntax:	`virtual ara::core::Result<std::size_t> ProcessBlock (WritableMemRegion out, 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
Parameters (out):	out	the output buffer
Return value:	ara::core::Result< std::size_t >	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	
Header file:	#include "ara/crypto/cryp/block_cryptor.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration.	
Notes:	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 (suppressPadding == true) the actual size of plain text should be equal to full size of the plain data block (defined by the algorithm)! [Error]: SecurityErrorDomain::kIncorrectInputSize if the mentioned above rules about the input size is violated [Error]: SecurityErrorDomain::kInsufficientCapacity if the out.size() is not enough to store the transformation result [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value	

⌋(*RS_CRYPTO_02311*)

8.15.1.1.2.7 template<typename Alloc = DefBytesAllocator> ara::core::Result<void> ara::crypto::cryp::BlockCryptor::ProcessBlock (ByteVectorT< Alloc > & *out*, ReadOnlyMemRegion *in*, bool *suppressPadding* = `false`) const `[noexcept]`

[SWS_CRYPT_20217]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BlockCryptor::ProcessBlock(ByteVectorT< Alloc > &out, ReadOnlyMemRegion in, bool suppressPadding=false)	
Scope:	class ara::crypto::cryp::BlockCryptor	
Syntax:	`template <typename Alloc> inline ara::core::Result<void> ProcessBlock (ByteVectorT< Alloc > &out, 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
Parameters (out):	out	the managed container for output block
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/block_cryptor.h"	
Description:	Process (encrypt / decrypt) an input block according to the cryptor configuration.	
Notes:	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() >= GetMaxOutputSize(false). Decryption expects what: in.size() == GetMaxInputSize() && out.capacity() >= GetMaxOutputSize(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)! [Error]: SecurityErrorDomain::kIncorrectInputSize if the mentioned above rules about the input size is violated [Error]: SecurityErrorDomain::kInsufficientCapacity if the out.size() is not enough to store the transformation result [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value	

⌋*(RS_CRYPTO_02311)*

8.15.1.2 class ara::crypto::cryp::BufferedDigest

[SWS_CRYPT_20300]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::BufferedDigest
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::StreamStarter

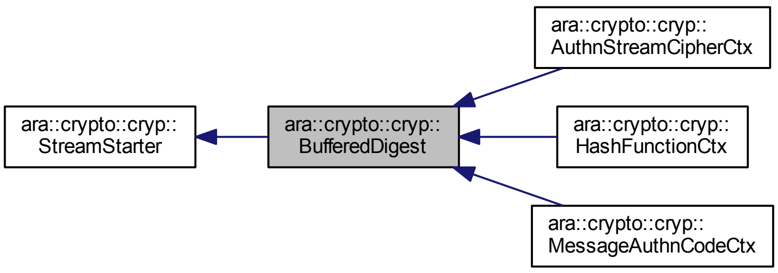
▽

△

Syntax:	`class BufferedDigest : public StreamStarter {...};`
Header file:	#include "ara/crypto/cryp/buffered_digest.h"
Description:	General interface for buffered computation of a digest (MAC/HMAC/hash).

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02205)*

Inheritance diagram for ara::crypto::cryp::BufferedDigest:



Public Member Functions

- virtual std::size_t GetDigestSize () const noexcept=0
- virtual ara::core::Result< void > Update (const KeyMaterial &in) noexcept=0
- virtual ara::core::Result< void > Update (ReadOnlyMemRegion in) noexcept=0
- virtual ara::core::Result< void > Update (Byte in) noexcept=0
- virtual ara::core::Result< Signature::Uptrc > Finish (bool makeSignatureObject=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- virtual ara::core::Result< std::size_t > GetDigest (WritableMemRegion output, std::size_t offset=0) const noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< void > GetDigest (ByteVectorT< Alloc > &output, std::size_t offset=0) const noexcept
- virtual ara::core::Result< bool > Compare (ReadOnlyMemRegion expected, std::size_t offset=0) const noexcept=0
- virtual ara::core::Result< bool > Check (const Signature &expected) const noexcept=0

Additional Inherited Members

8.15.1.2.1 Member Function Documentation

8.15.1.2.1.1 virtual std::size_t ara::crypto::cryp::BufferedDigest::GetDigestSize ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20311]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::GetDigestSize()	
Scope:	class ara::crypto::cryp::BufferedDigest	
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/buffered_digest.h"	
Description:	Get the output digest size.	

⌋*(RS_CRYPTO_02309)*

8.15.1.2.1.2 virtual ara::core::Result<void> ara::crypto::cryp::BufferedDigest::Update (const KeyMaterial & *in*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20312]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::Update(const KeyMaterial &in)	
Scope:	class ara::crypto::cryp::BufferedDigest	
Syntax:	`virtual ara::core::Result<void> Update (const KeyMaterial &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	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Update the digest calculation context by a new part of the message.	
Notes:	This method is dedicated for cases then the KeyMaterial is a part of the "message". [Error]: SecurityErrorDomain::kProcessingNotStarted if the digest calculation was not initiated by a call of the Start() method	

⌋*(RS_CRYPTO_02302)*

8.15.1.2.1.3 virtual ara::core::Result<void> ara::crypto::cryp::BufferedDigest::Update (ReadOnlyMemRegion *in*) `[pure virtual], [noexcept]`

[SWS_CRYPT_20313]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::Update(ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::BufferedDigest	
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	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Update the digest calculation context by a new part of the message.	
Notes:	[Error]: SecurityErrorDomain::kProcessingNotStarted if the digest calculation was not initiated by a call of the Start() method	

⌋*(RS_CRYPTO_02302)*

8.15.1.2.1.4 virtual ara::core::Result<void> ara::crypto::cryp::BufferedDigest::Update (Byte *in*) `[pure virtual], [noexcept]`

[SWS_CRYPT_20314]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::Update(Byte in)	
Scope:	class ara::crypto::cryp::BufferedDigest	
Syntax:	`virtual ara::core::Result<void> Update (Byte 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	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Update the digest calculation context by a new part of the message.	
Notes:	This method is convenient for processing of constant tags. [Error]: SecurityErrorDomain::kProcessingNotStarted if the digest calculation was not initiated by a call of the Start() method	

⌋*(RS_CRYPTO_02302)*

8.15.1.2.1.5 virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::BufferedDigest::Finish (bool *makeSignatureObject* = `false`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20315]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::Finish(bool makeSignatureObject=false, ReservedObject Index reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::BufferedDigest	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> Finish (bool makeSignature Object=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0;`	
Parameters (in):	makeSignatureObject	if this argument is true then the method will also produce the signature object
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Signature::Uptrc >	unique smart pointer to created signature object, if (makeSignatureObject == true) or nullptr if (make SignatureObject == false)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Finish the digest calculation and optionally produce the "signature" object.	
Notes:	Only after call of this method the digest can be signed, verified, extracted or compared! If the signature object produced by a plain hash-function then the dependence COUID of the "signature" should be set to COUID of domain parameters used by this context, but the "hash algorithm ID" field of the "signature" should be set according to own algorithm ID (i.e. equal to CryptoPrimitiveId::GetPrimitiveId()). 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, but the "hash algorithm ID" field of the "signature" should be set to kAlgIdNone (0). [Error]: SecurityErrorDomain::kProcessingNotStarted if the digest calculation was not initiated by a call of the Start() method [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of specified slot is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kUsageViolation if the buffered digest belongs to a MAC/HMAC context initialized by a key without kAllowSignature permission, but (makeSignatureObject == true)	

⌋*(RS_CRYPTO_02302, RS_CRYPTO_02404)*

8.15.1.2.1.6 virtual ara::core::Result<std::size_t> ara::crypto::cryp::BufferedDigest::GetDigest (WritableMemRegion *output*, std::size_t *offset = 0*) const [pure virtual], [noexcept]

[SWS_CRYPT_20316]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::GetDigest(WritableMemRegion output, std::size_t offset=0)	
Scope:	class ara::crypto::cryp::BufferedDigest	
Syntax:	`virtual ara::core::Result<std::size_t> GetDigest (WritableMemRegion output, 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
Parameters (out):	output	an output buffer for storing the requested digest fragment (or fully)
Return value:	ara::core::Result< std::size_t >	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	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Get requested part of calculated digest to existing memory buffer.	
Notes:	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. [Error]: SecurityErrorDomain::kProcessingNotFinished if the digest calculation was not finished by a call of the Finish() method [Error]: SecurityErrorDomain::kUsageViolation if the buffered digest is part of MAC/HMAC context initialized by a key without kAllowSignature permission	

⌋*(RS_CRYPTO_02404)*

8.15.1.2.1.7 template<typename Alloc = DefBytesAllocator> ara::core::Result<void> ara::crypto::cryp::BufferedDigest::GetDigest (ByteVectorT< Alloc > & *output*, std::size_t *offset = 0*) const [noexcept]

[SWS_CRYPT_20317]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::BufferedDigest::GetDigest(ByteVectorT< Alloc > &output, std::size_t offset=0)
Scope:	class ara::crypto::cryp::BufferedDigest

Syntax:	`template <typename Alloc>`
	`inline ara::core::Result<void> GetDigest (ByteVectorT< Alloc > &output, 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
Parameters (out):	output	a managed container for storing the requested digest fragment (or fully)
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Get requested part of calculated digest to pre-reserved managed container.	
Notes:	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. [Error]: SecurityErrorDomain::kProcessingNotFinished if the digest calculation was not finished by a call of the Finish() method [Error]: SecurityErrorDomain::kUsageViolation if the buffered digest is part of MAC/HMAC context initialized by a key without kAllowSignature permission	

⌋*(RS_CRYPTO_02404)*

8.15.1.2.1.8 virtual ara::core::Result<bool> ara::crypto::cryp::BufferedDigest::Compare (ReadOnlyMemRegion *expected*, std::size_t *offset* = *0*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20318]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::Compare(ReadOnlyMemRegion expected, std::size_t offset=0)	
Scope:	class ara::crypto::cryp::BufferedDigest	
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	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Compare the calculated digest against an expected value.	

▽

△

Notes:	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. [Error]: SecurityErrorDomain::kProcessingNotFinished if the digest calculation was not finished by a call of the Finish() method [Error]: SecurityErrorDomain::kBruteForceRisk if the buffered digest is a part of MAC/HMAC 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)

⌋*(RS_CRYPTO_02404)*

8.15.1.2.1.9 virtual ara::core::Result<bool> ara::crypto::cryp::BufferedDigest::Check (const Signature & *expected*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20319]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::BufferedDigest::Check(const Signature &expected)	
Scope:	class ara::crypto::cryp::BufferedDigest	
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	
Header file:	#include "ara/crypto/cryp/buffered_digest.h"	
Description:	Check the calculated digest against an expected "signature" object.	
Notes:	Entire digest value is kept in the context up to next call Start(), therefore it can be verified again or extracted. [Error]: SecurityErrorDomain::kProcessingNotFinished if the digest calculation was not finished by a call of the Finish() method [Error]: SecurityErrorDomain::kIncompatibleObject if the provided "signature" object was produced by another crypto primitive type	

⌋*(RS_CRYPTO_02203, RS_CRYPTO_02204)*

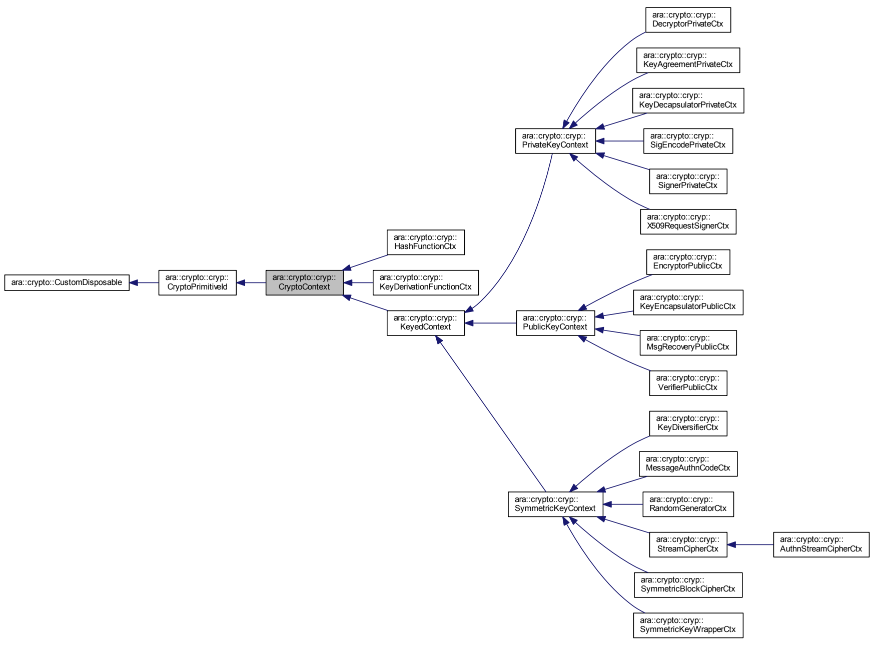
8.15.1.3 class ara::crypto::cryp::CryptoContext

[SWS_CRYPT_20400]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::CryptoContext
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::CryptoPrimitiveId
Syntax:	`class CryptoContext : public CryptoPrimitiveId {...};`
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_02311)*

Inheritance diagram for ara::crypto::cryp::CryptoContext:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::HashFunctionCtx
ara::crypto::cryp::KeyDerivationFunctionCtx
ara::crypto::cryp::KeyedContext
ara::crypto::cryp::PrivateKeyContext
ara::crypto::cryp::DecryptorPrivateCtx
ara::crypto::cryp::KeyAgreementPrivateCtx
ara::crypto::cryp::KeyDecapsulatorPrivateCtx
ara::crypto::cryp::SigEncodePrivateCtx
ara::crypto::cryp::SignerPrivateCtx
ara::crypto::cryp::X509RequestSignerCtx
ara::crypto::cryp::PublicKeyContext
ara::crypto::cryp::EncryptorPublicCtx
ara::crypto::cryp::KeyEncapsulatorPublicCtx
ara::crypto::cryp::MsgRecoveryPublicCtx
ara::crypto::cryp::VerifierPublicCtx
ara::crypto::cryp::SymmetricKeyContext
ara::crypto::cryp::KeyDiversifierCtx
ara::crypto::cryp::MessageAuthnCodeCtx
ara::crypto::cryp::RandomGeneratorCtx
ara::crypto::cryp::StreamCipherCtx
ara::crypto::cryp::AuthnStreamCipherCtx
ara::crypto::cryp::SymmetricBlockCipherCtx
ara::crypto::cryp::SymmetricKeyWrapperCtx

Public Member Functions

- virtual bool IsKeyedContext () const noexcept=0
- virtual bool IsInitialized () const noexcept=0
- virtual bool GetParametersUid (CryptoObjectUid ∗parametersUid=nullptr) const noexcept=0
- virtual ara::core::Result< void > Reset (DomainParameters::Sptrc params=nullptr) noexcept=0

Additional Inherited Members

8.15.1.3.1 Member Function Documentation

8.15.1.3.1.1 virtual bool ara::crypto::cryp::CryptoContext::IsKeyedContext ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_20411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoContext::IsKeyedContext()	
Scope:	class ara::crypto::cryp::CryptoContext	
Syntax:	`virtual bool IsKeyedContext () const noexcept=0;`	
Return value:	bool	true if the crypto context requires initialization by a key value
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Description:	Check if the crypto context requires initialization by a key value.	

⌋*(RS_CRYPTO_02309)*

8.15.1.3.1.2 virtual bool ara::crypto::cryp::CryptoContext::IsInitialized ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_20412]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoContext::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: domain parameters, key value, IV/seed, etc.	

⌋*(RS_CRYPTO_02309)*

8.15.1.3.1.3 virtual bool ara::crypto::cryp::CryptoContext::GetParametersUid (CryptoObjectUid * *parametersUid* = `nullptr`) const [pure virtual], [noexcept]

[SWS_CRYPT_20413]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoContext::GetParametersUid(CryptoObjectUid *parametersUid=nullptr)	
Scope:	class ara::crypto::cryp::CryptoContext	
Syntax:	`virtual bool GetParametersUid (CryptoObjectUid *parametersUid=nullptr) const noexcept=0;`	
Parameters (out):	parametersUid	optional pointer to the output buffer for the target COUID saving
Return value:	bool	true if the context has assigned domain parameters object and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Description:	Get the COUID of a domain parameters object assigned to the context.	
Notes:	Domain parameters always has type CryptoObjectType::kDomainParameters.	

⌋*(RS_CRYPTO_02005)*

8.15.1.3.1.4 virtual ara::core::Result<void> ara::crypto::cryp::CryptoContext::Reset (DomainParameters::Sptrc *params* = `nullptr`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20414]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoContext::Reset(DomainParameters::Sptrc params=nullptr)	
Scope:	class ara::crypto::cryp::CryptoContext	
Syntax:	`virtual ara::core::Result<void> Reset (DomainParameters::Sptrc params=nullptr) noexcept=0;`	
Parameters (in):	params	an optional shared pointer to a domain parameters object, which should be set to this crypto context
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_context.h"	
Description:	Clear the crypto context and set the domain parameters to it.	
Notes:	If domain parameters are principally required for a crypto algorithm assigned to the context then they must be set before loading of a key value (if it is a keyed context)! If (params == nullptr) then this method only clears the crypto context. [Error]: SecurityErrorDomain::kIncompatibleObject if provided domain parameters object is incompatible with an algorithm assigned to this cryptographic context [Error]: SecurityErrorDomain::kIncompleteArgState if provided domain parameters object has incomplete state	

⌋*(RS_CRYPTO_02108)*

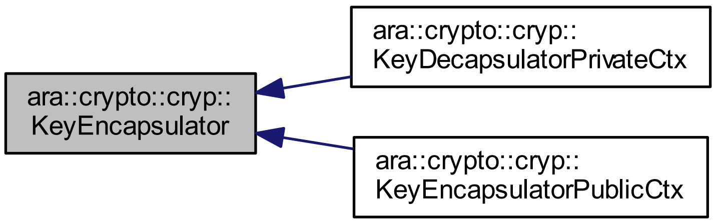
8.15.1.4 class ara::crypto::cryp::KeyEncapsulator

[SWS_CRYPT_21900]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyEncapsulator
Scope:	namespace ara::crypto::cryp
Syntax:	`class KeyEncapsulator {...};`
Header file:	#include "ara/crypto/cryp/key_encapsulator.h"
Description:	Basic interface of Asymmetric Key Encapsulation Mechanism (KEM).

⌋*(RS_CRYPTO_02209)*

Inheritance diagram for ara::crypto::cryp::KeyEncapsulator:



Public Member Functions

- virtual std::size_t GetKekEntropy () const noexcept=0
- virtual std::size_t GetEncapsulatedSize () const noexcept=0

Protected Member Functions

- virtual ∼KeyEncapsulator ()=default

8.15.1.4.1 Constructor & Destructor Documentation

8.15.1.4.1.1 virtual ara::crypto::cryp::KeyEncapsulator::∼KeyEncapsulator ( ) [protected], [virtual], [default]

[SWS_CRYPT_21910]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::KeyEncapsulator::~KeyEncapsulator()
Scope:	class ara::crypto::cryp::KeyEncapsulator
Visibility:	protected
Syntax:	`virtual ~KeyEncapsulator ()=default;`
Header file:	#include "ara/crypto/cryp/key_encapsulator.h"
Description:	Virtual destructor.

⌋*(RS_CRYPTO_02311)*

8.15.1.4.2 Member Function Documentation

8.15.1.4.2.1 virtual std::size_t ara::crypto::cryp::KeyEncapsulator::GetKekEntropy ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_21911]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyEncapsulator::GetKekEntropy()	
Scope:	class ara::crypto::cryp::KeyEncapsulator	
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.h"	
Description:	Get entropy (bit-length) of the key encryption key (KEK) material.	
Notes:	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)*

8.15.1.4.2.2 virtual std::size_t ara::crypto::cryp::KeyEncapsulator::GetEncapsulatedSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_21912]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::KeyEncapsulator::GetEncapsulatedSize()
Scope:	class ara::crypto::cryp::KeyEncapsulator
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.h"	
Description:	Get fixed size of the encapsulated data block.	

⌋(*RS_CRYPTO_02309*)

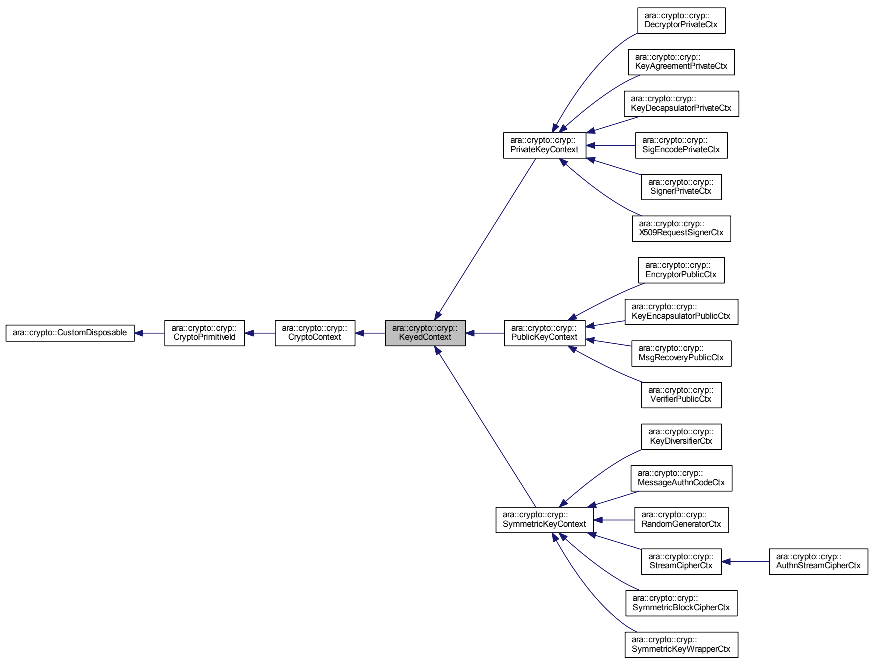
8.15.1.5 class ara::crypto::cryp::KeyedContext

[SWS_CRYPT_21700]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyedContext
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::CryptoContext
Syntax:	`class KeyedContext : public CryptoContext {...};`
Header file:	#include "ara/crypto/cryp/keyed_context.h"
Description:	A common interface of all keyed cryptographic contextes.

⌋(*RS_CRYPTO_02311*)

Inheritance diagram for ara::crypto::cryp::KeyedContext:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::KeyedContext
ara::crypto::cryp::PrivateKeyContext
ara::crypto::cryp::DecryptorPrivateCtx
ara::crypto::cryp::KeyAgreementPrivateCtx
ara::crypto::cryp::KeyDecapsulatorPrivateCtx
ara::crypto::cryp::SigEncodePrivateCtx
ara::crypto::cryp::SignerPrivateCtx
ara::crypto::cryp::X509RequestSignerCtx
ara::crypto::cryp::PublicKeyContext
ara::crypto::cryp::EncryptorPublicCtx
ara::crypto::cryp::KeyEncapsulatorPublicCtx
ara::crypto::cryp::MsgRecoveryPublicCtx
ara::crypto::cryp::VerifierPublicCtx
ara::crypto::cryp::SymmetricKeyContext
ara::crypto::cryp::KeyDiversifierCtx
ara::crypto::cryp::MessageAuthnCodeCtx
ara::crypto::cryp::RandomGeneratorCtx
ara::crypto::cryp::StreamCipherCtx
ara::crypto::cryp::AuthnStreamCipherCtx
ara::crypto::cryp::SymmetricBlockCipherCtx
ara::crypto::cryp::SymmetricKeyWrapperCtx

Public Member Functions

- virtual bool IsKeyBitLengthSupported (std::size_t keyBitLength) const noexcept=0
- virtual std::size_t GetMinKeyBitLength () const noexcept=0
- virtual std::size_t GetMaxKeyBitLength () const noexcept=0
- virtual std::size_t GetActualKeyBitLength (CryptoObjectUid ∗keyId=nullptr) const noexcept=0

Additional Inherited Members

8.15.1.5.1 Member Function Documentation

8.15.1.5.1.1 virtual bool ara::crypto::cryp::KeyedContext::IsKeyBitLengthSupported (std::size_t *keyBitLength*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21711]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyedContext::IsKeyBitLengthSupported(std::size_t keyBitLength)	
Scope:	class ara::crypto::cryp::KeyedContext	
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/keyed_context.h"	
Description:	Verify supportness of specific key length by the context.	

⌋*(RS_CRYPTO_02309)*

8.15.1.5.1.2 virtual std::size_t ara::crypto::cryp::KeyedContext::GetMinKeyBitLength ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21712]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyedContext::GetMinKeyBitLength()	
Scope:	class ara::crypto::cryp::KeyedContext	
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/keyed_context.h"	
Description:	Get minimal supported key length in bits.	

⌋*(RS_CRYPTO_02309)*

8.15.1.5.1.3 virtual std::size_t ara::crypto::cryp::KeyedContext::GetMaxKeyBitLength ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_21713]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyedContext::GetMaxKeyBitLength()	
Scope:	class ara::crypto::cryp::KeyedContext	
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/keyed_context.h"	
Description:	Get maximal supported key length in bits.	

⌋*(RS_CRYPTO_02309)*

8.15.1.5.1.4 virtual std::size_t ara::crypto::cryp::KeyedContext::GetActualKeyBitLength (CryptoObjectUid ∗ *keyId* = nullptr) const [pure virtual], [noexcept]

[SWS_CRYPT_21714]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::KeyedContext::GetActualKeyBitLength(CryptoObjectUid *keyId=nullptr)
Scope:	class ara::crypto::cryp::KeyedContext
Syntax:	`virtual std::size_t GetActualKeyBitLength (CryptoObjectUid *key Id=nullptr) const noexcept=0;`

▽

△

Parameters (out):	keyId	optional pointer to a buffer for saving an COUID of a key now loaded to the context
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/keyed_context.h"	
Description:	Get actual bit-length of a key loaded to the context.	
Notes:	If any key was not set to the context yet then 0 is returned.	

⌋*(RS_CRYPTO_02309)*

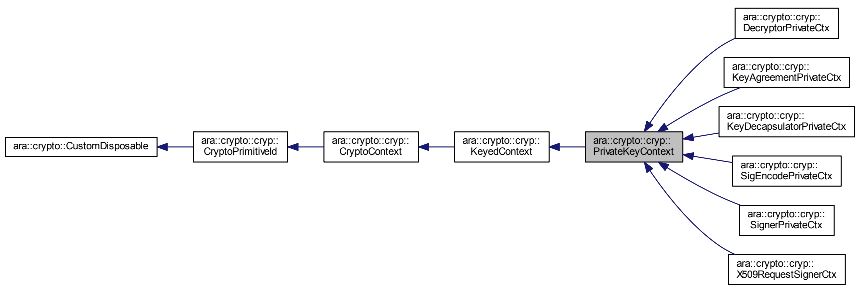
8.15.1.6 class ara::crypto::cryp::PrivateKeyContext

[SWS_CRYPT_22600]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::PrivateKeyContext
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::KeyedContext
Syntax:	`class PrivateKeyContext : public KeyedContext {...};`
Header file:	#include "ara/crypto/cryp/private_key_context.h"
Description:	Generalized interface of Private Key algorithm Context.

⌋*(RS_CRYPTO_02202)*

Inheritance diagram for ara::crypto::cryp::PrivateKeyContext:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: PrivateKeyContext
ara::crypto::cryp:: DecryptorPrivateCtx
ara::crypto::cryp:: KeyAgreementPrivateCtx
ara::crypto::cryp:: KeyDecapsulatorPrivateCtx
ara::crypto::cryp:: SigEncodePrivateCtx
ara::crypto::cryp:: SignerPrivateCtx
ara::crypto::cryp:: X509RequestSignerCtx

Public Member Functions

- virtual ara::core::Result< void > SetKey (const PrivateKey &key) noexcept=0

Additional Inherited Members

8.15.1.6.1 Member Function Documentation

8.15.1.6.1.1 virtual ara::core::Result<void> ara::crypto::cryp::PrivateKeyContext::SetKey (const PrivateKey & *key*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22611]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PrivateKeyContext::SetKey(const PrivateKey &key)	
Scope:	class ara::crypto::cryp::PrivateKeyContext	
Syntax:	`virtual ara::core::Result<void> SetKey (const PrivateKey &key) noexcept=0;`	
Parameters (in):	key	a reference to a source key
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/private_key_context.h"	
Description:	Set (load) a key to the algorithm context.	
Notes:	If domain parameters are principally required for a crypto algorithm assigned to this context then they must be set before the loading of a key value (via call of the CryptoContext::Reset())! [Error]: SecurityErrorDomain::kIncompatibleObject if provided key object is incompatible with an algorithm assigned to this private key context [Error]: SecurityErrorDomain::kBadObjectReference if provided private key object references to an instance of the domain parameters different from the one loaded to the context, i.e. if the COUID of the domain parameters in the context (see GetParametersUid()) is not equal to the COUID referenced from the private key object [Error]: SecurityErrorDomain::kUsageViolation if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object	

⌋*(RS_CRYPTO_02002, RS_CRYPTO_02003, RS_CRYPTO_02108)*

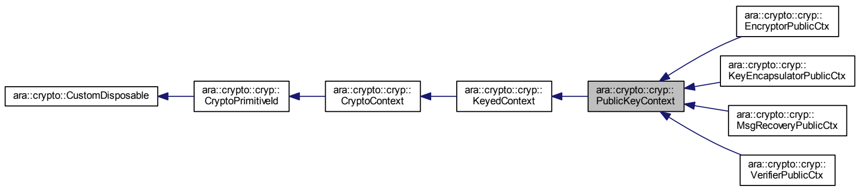
8.15.1.7 class ara::crypto::cryp::PublicKeyContext

[SWS_CRYPT_22800]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::PublicKeyContext
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::KeyedContext
Syntax:	`class PublicKeyContext : public KeyedContext {...};`
Header file:	#include "ara/crypto/cryp/public_key_context.h"
Description:	Generalized interface of Public Key algorithm Context.

⌋*(RS_CRYPTO_02202)*

Inheritance diagram for ara::crypto::cryp::PublicKeyContext:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::KeyedContext
ara::crypto::cryp::PublicKeyContext
ara::crypto::cryp::EncryptorPublicCtx
ara::crypto::cryp::KeyEncapsulatorPublicCtx
ara::crypto::cryp::MsgRecoveryPublicCtx
ara::crypto::cryp::VerifierPublicCtx

Public Member Functions

- virtual ara::core::Result< void > SetKey (const PublicKey &key) noexcept=0

Additional Inherited Members

8.15.1.7.1 Member Function Documentation

8.15.1.7.1.1 virtual ara::core::Result<void> ara::crypto::cryp::PublicKeyContext::SetKey (const PublicKey & *key*) `[pure virtual], [noexcept]`

[SWS_CRYPT_22811]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PublicKeyContext::SetKey(const PublicKey &key)	
Scope:	class ara::crypto::cryp::PublicKeyContext	
Syntax:	`virtual ara::core::Result<void> SetKey (const PublicKey &key) noexcept=0;`	
Parameters (in):	key	a reference to a source key
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/public_key_context.h"	
Description:	Set (load) a key to the algorithm context.	
Notes:	If domain parameters are principally required for a crypto algorithm assigned to this context then they must be set before the loading of a key value (via call of the CryptoContext::Reset())! [Error]: SecurityErrorDomain::kIncompatibleObject if provided key object is incompatible with an algorithm assigned to this public key context [Error]: SecurityErrorDomain::kBadObjectReference if provided public key object references to an instance of the domain parameters different from the one loaded to the context, i.e. if the ▽	

▽

△

	△ COUID of the domain parameters in the context (see GetParametersUid()) is not equal to the COUID referenced from the public key object [Error]: SecurityErrorDomain::kUsageViolation if the transformation type associated with this context is prohibited by the "allowed usage" restrictions of provided key object

⌋*(RS_CRYPTO_02108)*

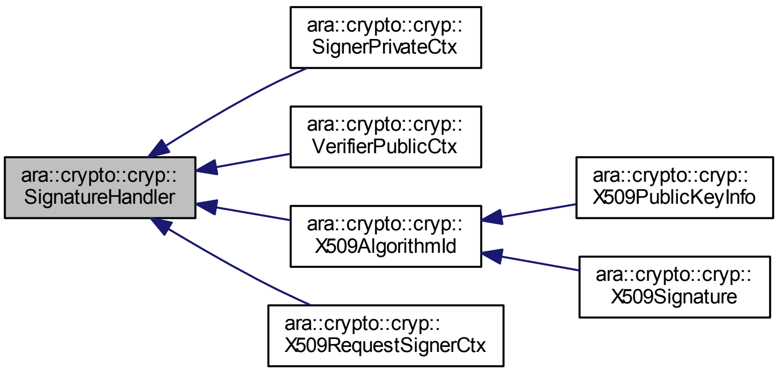
8.15.1.8 class ara::crypto::cryp::SignatureHandler

[SWS_CRYPT_23400]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::SignatureHandler
Scope:	namespace ara::crypto::cryp
Syntax:	`class SignatureHandler {...};`
Header file:	#include "ara/crypto/cryp/signature_handler.h"
Description:	A basic interface for both types of the signature handlers: signer and verifier.

⌋*(RS_CRYPTO_02204)*

Inheritance diagram for ara::crypto::cryp::SignatureHandler:



Public Member Functions

- virtual std::size_t GetRequiredHashSize () const noexcept=0
- virtual CryptoPrimitiveId::AlgId GetRequiredHashAlgId () const noexcept=0
- virtual std::size_t GetSignatureSize () const noexcept=0

Protected Member Functions

- virtual ∼SignatureHandler () noexcept=default

8.15.1.8.1 Constructor & Destructor Documentation

8.15.1.8.1.1 virtual ara::crypto::cryp::SignatureHandler::∼SignatureHandler ( ) [protected], [virtual], [default], [noexcept]

[SWS_CRYPT_23410]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02311)*

8.15.1.8.2 Member Function Documentation

8.15.1.8.2.1 virtual std::size_t ara::crypto::cryp::SignatureHandler::GetRequiredHashSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_23411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SignatureHandler::GetRequiredHashSize()	
Scope:	class ara::crypto::cryp::SignatureHandler	
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_handler.h"	
Description:	Get the hash size required by current signature algorithm.	

⌋*(RS_CRYPTO_02309)*

8.15.1.8.2.2 virtual CryptoPrimitiveId::AlgId ara::crypto::cryp::SignatureHandler::GetRequiredHashAlgId ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23412]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SignatureHandler::GetRequiredHashAlgId()	
Scope:	class ara::crypto::cryp::SignatureHandler	
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_handler.h"	
Description:	Get an ID of hash algorithm required by current signature algorithm.	

⌋*(RS_CRYPTO_02309)*

8.15.1.8.2.3 virtual std::size_t ara::crypto::cryp::SignatureHandler::GetSignatureSize ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23413]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SignatureHandler::GetSignatureSize()	
Scope:	class ara::crypto::cryp::SignatureHandler	
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_handler.h"	
Description:	Get size of the signature value produced and required by the current algorithm.	

⌋*(RS_CRYPTO_02309)*

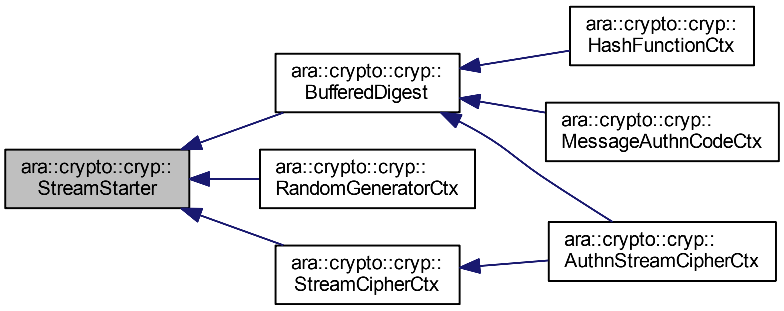
8.15.1.9 class ara::crypto::cryp::StreamStarter

[SWS_CRYPT_24700]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::StreamStarter
Scope:	namespace ara::crypto::cryp
Syntax:	`class StreamStarter {...};`
Header file:	#include "ara/crypto/cryp/stream_starter.h"
Description:	Starter interface of a stream processing, common for all primitives that supports the streamable processing approach.

⌋*(RS_CRYPTO_02302)*

Inheritance diagram for ara::crypto::cryp::StreamStarter:


ara::crypto::cryp:: StreamStarter
ara::crypto::cryp:: BufferedDigest
ara::crypto::cryp:: RandomGeneratorCtx
ara::crypto::cryp:: StreamCipherCtx
ara::crypto::cryp:: HashFunctionCtx
ara::crypto::cryp:: MessageAuthnCodeCtx
ara::crypto::cryp:: AuthnStreamCipherCtx

Public Member Functions

- virtual std::size_t GetIvSize () const noexcept=0
- virtual std::size_t GetBlockSize () const noexcept=0
- virtual bool IsValidIvSize (std::size_t ivSize) const noexcept=0
- virtual ara::core::Result< void > Start (ReadOnlyMemRegion iv=ReadOnlyMemRegion()) noexcept=0
- virtual ara::core::Result< void > Start (const SecretSeed &iv) noexcept=0
- virtual std::size_t GetActualIvBitLength (CryptoObjectUid ∗ivUid=nullptr) const noexcept=0

Protected Member Functions

- virtual ∼StreamStarter () noexcept=default

8.15.1.9.1 Constructor & Destructor Documentation

8.15.1.9.1.1 virtual ara::crypto::cryp::StreamStarter::∼StreamStarter ( ) [protected], [virtual], [default], [noexcept]

[SWS_CRYPT_24710]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02311)*

8.15.1.9.2 Member Function Documentation

8.15.1.9.2.1 virtual std::size_t ara::crypto::cryp::StreamStarter::GetIvSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_24711]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamStarter::GetIvSize()	
Scope:	class ara::crypto::cryp::StreamStarter	
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/stream_starter.h"	
Description:	Get default expected size of the Initialization Vector (IV) or nonce.	

⌋*(RS_CRYPTO_02309)*

8.15.1.9.2.2 virtual std::size_t ara::crypto::cryp::StreamStarter::GetBlockSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_24712]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamStarter::GetBlockSize()	
Scope:	class ara::crypto::cryp::StreamStarter	
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/stream_starter.h"	
Description:	Get block (or internal buffer) size of the base algorithm.	
Notes:	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)*

8.15.1.9.2.3 virtual bool ara::crypto::cryp::StreamStarter::IsValidIvSize (std::size_t *ivSize*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24713]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamStarter::IsValidIvSize(std::size_t ivSize)	
Scope:	class ara::crypto::cryp::StreamStarter	
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/stream_starter.h"	
Description:	Verify validity of specific Initialization Vector (IV) length.	

⌋*(RS_CRYPTO_02309)*

8.15.1.9.2.4 virtual ara::core::Result<void> ara::crypto::cryp::StreamStarter::Start (ReadOnlyMemRegion *iv* = ReadOnlyMemRegion *()*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24714]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamStarter::Start(ReadOnlyMemRegion iv=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::StreamStarter	
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	
Header file:	#include "ara/crypto/cryp/stream_starter.h"	
Description:	Initialize the context for a new data stream processing or generation (depending from the primitive).	
Notes:	[Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by required domain parameters or key object [Error]: SecurityErrorDomain::kInvalidInputSize if the size of provided IV is not supported (i.e. if it is not enough for the initialization) [Error]: 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) 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*)

8.15.1.9.2.5 virtual ara::core::Result<void> ara::crypto::cryp::StreamStarter::Start (const SecretSeed & *iv*) `[pure virtual], [noexcept]`

[SWS_CRYPT_24715]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamStarter::Start(const SecretSeed &iv)	
Scope:	class ara::crypto::cryp::StreamStarter	
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	
Header file:	#include "ara/crypto/cryp/stream_starter.h"	
Description:	Initialize the context for a new data stream processing or generation (depending from the primitive).	
Notes:	[Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by required domain parameters or key object [Error]: SecurityErrorDomain::kInvalidInputSize if the size of provided IV is not supported (i.e. if it is not enough for the initialization) ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kUnsupported if the base algorithm (or its current implementation) principally doesn't support the IV variation [Error]: SecurityErrorDomain::kUsageViolation if this transformation type is prohibited by the "allowed usage" restrictions of the provided SecretSeed object 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*)

8.15.1.9.2.6 virtual std::size_t ara::crypto::cryp::StreamStarter::GetActualIvBitLength (CryptoObjectUid ∗ *ivUid* = `nullptr`) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24716]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamStarter::GetActualIvBitLength(CryptoObjectUid *ivUid=nullptr)	
Scope:	class ara::crypto::cryp::StreamStarter	
Syntax:	`virtual std::size_t GetActualIvBitLength (CryptoObjectUid *ivUid=nullptr) const noexcept=0;`	
Parameters (out):	ivUid	optional pointer to a buffer for saving an COUID of a IV object now loaded to the context
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/stream_starter.h"	
Description:	Get actual bit-length of an IV loaded to the context.	
Notes:	If any IV was not set to the context yet then 0 is returned. 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.	

⌋(*RS_CRYPTO_02309*)

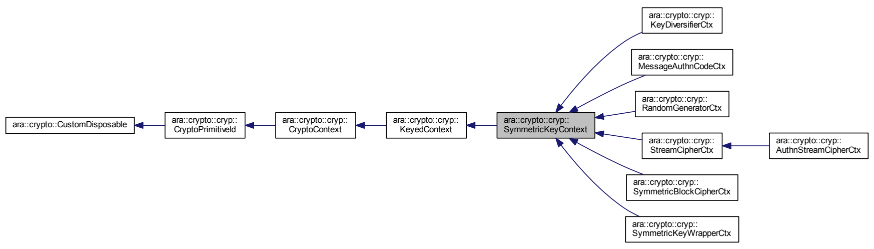
8.15.1.10 class ara::crypto::cryp::SymmetricKeyContext

[SWS_CRYPT_23900]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::SymmetricKeyContext
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::KeyedContext
Syntax:	`class SymmetricKeyContext : public KeyedContext {...};`
Header file:	#include "ara/crypto/cryp/symmetric_key_context.h"
Description:	Generalized interface of Symmetric Key algorithm Context.

⌋*(RS_CRYPTO_02201)*

Inheritance diagram for ara::crypto::cryp::SymmetricKeyContext:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: SymmetricKeyContext
ara::crypto::cryp:: KeyDiversifierCtx
ara::crypto::cryp:: MessageAuthnCodeCtx
ara::crypto::cryp:: RandomGeneratorCtx
ara::crypto::cryp:: StreamCipherCtx
ara::crypto::cryp:: AuthnStreamCipherCtx
ara::crypto::cryp:: SymmetricBlockCipherCtx
ara::crypto::cryp:: SymmetricKeyWrapperCtx

Public Member Functions

- virtual ara::core::Result< void > SetKey (const SymmetricKey &key, bool directTransform=true) noexcept=0
- virtual bool IsDirectTransform () const noexcept=0

Additional Inherited Members

8.15.1.10.1 Member Function Documentation

8.15.1.10.1.1 virtual ara::core::Result<void> ara::crypto::cryp::SymmetricKeyContext::SetKey (const SymmetricKey & *key*, bool *directTransform* = `true`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23911]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::SymmetricKeyContext::SetKey(const SymmetricKey &key, bool direct Transform=true)

▽

△

Scope:	class ara::crypto::cryp::SymmetricKeyContext	
Syntax:	`virtual ara::core::Result<void> SetKey (const SymmetricKey &key, bool directTransform=true) noexcept=0;`	
Parameters (in):	key	the source key object
	directTransform	the "direction" indicator: deploy the key for direct transformation (if true) or for reverse one (if false)
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key_context.h"	
Description:	Set (deploy) a key to the symmetric algorithm context.	
Notes:	If domain parameters are principally required for a crypto algorithm assigned to this context then they must be set before the loading of a key value (via call of the CryptoContext::Reset())! [Error]: SecurityErrorDomain::kIncompatibleObject if the crypto primitive of the provided key object is incompatible with this symmetric key context [Error]: SecurityErrorDomain::kUsageViolation if the transformation type associated with this context (taking into account the direction specified by directTransform) is prohibited by the "allowed usage" restrictions of provided key object	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02003)*

8.15.1.10.1.2 virtual bool ara::crypto::cryp::SymmetricKeyContext::IsDirectTransform ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_23912]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyContext::IsDirectTransform()	
Scope:	class ara::crypto::cryp::SymmetricKeyContext	
Syntax:	`virtual bool IsDirectTransform () const noexcept=0;`	
Return value:	bool	true if the context configured for a direct transformation and false if for a reverse one
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key_context.h"	
Description:	Get configured "direction" of the transformation: direct (e.g. encryption) or reverse (e.g. decryption)	

⌋*(RS_CRYPTO_02309)*

8.16 Top-level cryptographic interfaces

Modules

- Top-level object interfaces
- Top-level transformation interfaces

Classes

- class ara::crypto::cryp::CryptoProvider

Detailed Description

This group consists of top-level cryptographic interfaces available for consumer applications via correspondent factory methods.

8.16.1 Class Documentation

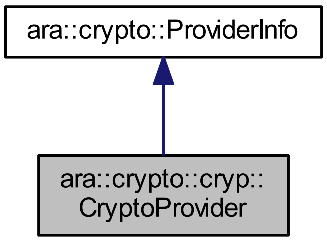
8.16.1.1 class ara::crypto::cryp::CryptoProvider

[SWS_CRYPT_20700]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::CryptoProvider
Scope:	namespace ara::crypto::cryp
Base class:	std::enable_shared_from_this< CryptoProvider >
Syntax:	`class CryptoProvider : public enable_shared_from_this< 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.
Notes:	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. Any user of this interface should create shared pointers to it only by calls of the method shared_from_this()!

⌋*(RS_CRYPTO_02305, RS_CRYPTO_02307, RS_CRYPTO_02401)*

Inheritance diagram for ara::crypto::cryp::CryptoProvider:



Public Types

- using Sptr = std::shared_ptr< CryptoProvider >
- using ObjectType = CryptoObject::Type
- using AlgId = CryptoPrimitiveId::AlgId
- using ContainedContextsList = std::vector< std::pair< AlgId, bool > >
- using ContainedObjectsList = std::vector< std::pair< AlgId, ObjectType > >
- using ContextReservationMap = std::vector< ContainedContextsList >
- using ObjectReservationMap = std::vector< ContainedObjectsList >

Public Member Functions

- virtual ∼CryptoProvider () noexcept=default
- virtual AlgId ConvertToAlgId (ara::core::StringView primitiveName) const noexcept=0
- virtual ara::core::Result< ara::core::String > ConvertToAlgName (AlgId algId) const noexcept=0
- virtual ara::core::Result< CryptoPrimitiveId::Category > GetPrimitiveCategory (AlgId algId) const noexcept=0
- virtual ara::core::Result< void > ReserveContexts (const ContextReservationMap &reservationMap) noexcept=0
- virtual ara::core::Result< void > ReserveObjects (const ObjectReservationMap &reservationMap) noexcept=0
- virtual ara::core::Result< void > ReserveContexts (std::size_t quantity) noexcept=0

- virtual ara::core::Result< void > ReserveObjects (std::size_t quantity) noexcept=0
- virtual void EnterRealTimeMode () noexcept=0
- virtual void LeaveRealTimeMode () noexcept=0
- virtual ara::core::Result< DomainParameters::Sptr > AllocDomainParameters (AlgId algId, bool isSession=false, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- virtual ara::core::Result< DomainParameters::Sptrc > KnownDomainParameters (ara::core::StringView oidName, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- virtual ara::core::Result< SymmetricKey::Uptrc > GenerateSymmetricKey (AlgId algId, Key::Usage allowedUsage, bool isSession=true, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- virtual ara::core::Result< PrivateKey::Uptrc > GeneratePrivateKey (AlgId algId, Key::Usage allowedUsage, bool isSession=false, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- virtual ara::core::Result< SecretSeed::Uptrc > GenerateSeed (AlgId algId, SecretSeed::Usage allowedUsage, bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- virtual ara::core::Result< std::size_t > GetSerializedSize (ObjectType objectType, AlgId algId, Serializable::FormatId formatId=Serializable::kFormatDefault) const noexcept=0
- virtual ara::core::Result< std::size_t > GetStorageSize (ObjectType objectType, AlgId algId) const noexcept=0
- virtual ara::core::Result< TrustedContainer::Uptr > AllocVolatileContainer (std::size_t capacity=0) noexcept=0
- virtual ara::core::Result< TrustedContainer::Uptr > AllocVolatileContainer (const ContainedObjectsList &objectsList) noexcept=0
- virtual ara::core::Result< std::size_t > ExportSecuredObject (const CryptoObject &object, SymmetricKeyWrapperCtx &transportContext, WritableMemRegion serialized=WritableMemRegion()) noexcept=0
- virtual ara::core::Result< std::size_t > ExportSecuredObject (const TrustedContainer &container, SymmetricKeyWrapperCtx &transportContext, WritableMemRegion serialized=WritableMemRegion()) noexcept=0
- virtual ara::core::Result< void > ImportSecuredObject (TrustedContainer &container, ReadOnlyMemRegion serialized, SymmetricKeyWrapperCtx &transportContext, bool isExportable=false, ObjectType expectedObject=ObjectType::kUnknown) noexcept=0

- virtual ara::core::Result< std::size_t > ExportPublicObject (const TrustedContainer &container, WritableMemRegion serialized=WritableMemRegion(), Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0
- virtual ara::core::Result< void > ImportPublicObject (TrustedContainer &container, ReadOnlyMemRegion serialized, ObjectType expectedObject=ObjectType::kUnknown) noexcept=0
- virtual ara::core::Result< CryptoObject::Uptrc > LoadObject (const TrustedContainer &container, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- template<typename ExpectedObject >
  ara::core::Result< typename ExpectedObject::Uptrc > LoadConcreteObject (const TrustedContainer &container, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept
- virtual ara::core::Result< PasswordCache::Uptr > AllocPasswordCache (std::size_t maximalLength, std::size_t requiredLength, unsigned requiredComplexity, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< PasswordHash::Uptr > HashPassword (HashFunctionCtx &hashCtx, const PasswordCache &password, bool isSession=false, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0
- virtual RandomGeneratorCtx::Sptr DefaultRng () noexcept=0
- virtual ara::core::Result< void > SetDefaultRng (RandomGeneratorCtx::Sptr rng=nullptr) noexcept=0
- virtual ara::core::Result< RandomGeneratorCtx::Sptr > CreateRandomGeneratorCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< SymmetricBlockCipherCtx::Uptr > CreateSymmetricBlockCipherCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< SymmetricKeyWrapperCtx::Uptr > CreateSymmetricKeyWrapperCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< StreamCipherCtx::Uptr > CreateStreamCipherCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< AuthnStreamCipherCtx::Uptr > CreateAuthnStreamCipherCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< MessageAuthnCodeCtx::Uptr > CreateMessageAuthnCodeCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0

- virtual ara::core::Result< HashFunctionCtx::Uptr > CreateHashFunctionCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< KeyDerivationFunctionCtx::Uptr > CreateKeyDerivationFunctionCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< KeyDiversifierCtx::Uptr > CreateKeyDiversifierCtx (AlgId masterAlgId, std::size_t slaveKeyLength, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< EncryptorPublicCtx::Uptr > CreateEncryptorPublicCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< DecryptorPrivateCtx::Uptr > CreateDecryptorPrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< KeyEncapsulatorPublicCtx::Uptr > CreateKeyEncapsulatorPublicCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< KeyDecapsulatorPrivateCtx::Uptr > CreateKeyDecapsulatorPrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< SigEncodePrivateCtx::Uptr > CreateSigEncodePrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< MsgRecoveryPublicCtx::Uptr > CreateMsgRecoveryPublicCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< SignerPrivateCtx::Uptr > CreateSignerPrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< VerifierPublicCtx::Uptr > CreateVerifierPublicCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< KeyAgreementPrivateCtx::Uptr > CreateKeyAgreementPrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0
- virtual ara::core::Result< X509RequestSignerCtx::Uptr > CreateX509RequestSignerCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0

Additional Inherited Members

8.16.1.1.1 Member Typedef Documentation

8.16.1.1.1.1 using ara::crypto::cryp::CryptoProvider::Sptr = std::shared_ptr<CryptoProvider>

[SWS_CRYPT_20701]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoProvider::Sptr
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	std::shared_ptr<CryptoProvider>
Syntax:	`using ara::crypto::cryp::CryptoProvider::Sptr = std::shared_ptr<CryptoProvider>;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.16.1.1.1.2 using ara::crypto::cryp::CryptoProvider::ObjectType = CryptoObject::Type

[SWS_CRYPT_20702]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoProvider::ObjectType
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	CryptoObject::Type
Syntax:	`using ara::crypto::cryp::CryptoProvider::ObjectType = CryptoObject::Type;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Enumeration of all types of crypto objects.

⌋*(RS_CRYPTO_02311)*

8.16.1.1.1.3 using ara::crypto::cryp::CryptoProvider::AlgId = CryptoPrimitiveId::AlgId

[SWS_CRYPT_20703]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoProvider::AlgId
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	CryptoPrimitiveId::AlgId
Syntax:	`using ara::crypto::cryp::CryptoProvider::AlgId = CryptoPrimitiveId::AlgId;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	A short alias for Algorithm ID type definition.

⌋*(RS_CRYPTO_02404)*

8.16.1.1.1.4 using ara::crypto::cryp::CryptoProvider::ContainedContextsList = std::vector<std::pair<AlgId, bool> >

[SWS_CRYPT_20704]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoProvider::ContainedContextsList
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	std::vector<std::pair<AlgId, bool> >
Syntax:	`using ara::crypto::cryp::CryptoProvider::ContainedContextsList = std::vector<std::pair<AlgId, bool> >;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	A list of Crypto Contexts that should be contained by a single shared memory slot in different time moments. This vector indirectly specifies a minimal required capacity of a single reserved Context slot via a list of Contexts' IDs that must be hosted by the slot. 1st element of the pair (type AlgId) specifies target Crypto Primitive via it's algorithm ID. 2nd element of the pair (type bool) specifies direct (true) or reverse (false) transformation.
Notes:	This vector is used for calculation of minimal required capacity of the reserved Context slot. If at least one element of AlgId has value 0 (kAlgIdUndefined), then maximal supported Context size should be reserved.

⌋*(RS_CRYPTO_02404)*

8.16.1.1.1.5 using ara::crypto::cryp::CryptoProvider::ContainedObjectsList = std::vector<std::pair<AlgId, ObjectType> >

[SWS_CRYPT_20705]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoProvider::ContainedObjectsList
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	std::vector<std::pair<AlgId, ObjectType> >
Syntax:	`using ara::crypto::cryp::CryptoProvider::ContainedObjectsList = std::vector<std::pair<AlgId, ObjectType> >;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	A list of Crypto Objects that should be contained by a single shared memory slot in different time moments. This vector indirectly specifies a minimal required capacity of a single reserved Object slot via a list of Objects' IDs that must be hosted by the slot. 1st element of the pair (type AlgId) specifies target Crypto Primitive via it's algorithm ID. 2nd element of the pair (type ObjectType) specifies concrete object type.
Notes:	This vector is used for calculation of minimal required capacity of the reserved Object slot. If some AlgId element of the list has value 0 (kAlgIdUndefined), then maximal supported size of correspondent ObjectType should be reserved for this element. If at least one element ObjectType of the list has value 0 (ObjectType::kUnknown), then maximal supported object size should be reserved.

⌋*(RS_CRYPTO_02404)*

8.16.1.1.1.6 using ara::crypto::cryp::CryptoProvider::ContextReservationMap = std::vector<ContainedContextsList>

[SWS_CRYPT_20706]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoProvider::ContextReservationMap
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	std::vector<ContainedContextsList>
Syntax:	`using ara::crypto::cryp::CryptoProvider::ContextReservationMap = std::vector<ContainedContextsList>;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	This vector specifies a whole mapping of minimally required capacities to the Context slots' indexes.

⌋*(RS_CRYPTO_02404)*

8.16.1.1.1.7 using ara::crypto::cryp::CryptoProvider::ObjectReservationMap = std::vector<ContainedObjectsList>

[SWS_CRYPT_20707]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::CryptoProvider::ObjectReservationMap
Scope:	class ara::crypto::cryp::CryptoProvider
Derived from:	std::vector<ContainedObjectsList>
Syntax:	`using ara::crypto::cryp::CryptoProvider::ObjectReservationMap = std::vector<ContainedObjectsList>;`
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	This vector specifies a whole mapping of minimally required capacities to the Object slots' indexes.

⌋*(RS_CRYPTO_02404)*

8.16.1.1.2 Constructor & Destructor Documentation

8.16.1.1.2.1 virtual ara::crypto::cryp::CryptoProvider::∼CryptoProvider ( ) [virtual], [default], [noexcept]

[SWS_CRYPT_20710]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoProvider::~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_02311)*

8.16.1.1.3 Member Function Documentation

8.16.1.1.3.1 virtual AlgId ara::crypto::cryp::CryptoProvider::ConvertToAlgId (ara::core::StringView *primitiveName*) const [pure virtual], [noexcept]

[SWS_CRYPT_20711]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoProvider::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 kAlgId Undefined if a primitive with provided name is not supported
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Convert a common name of crypto algorithm to a correspondent vendor specific binary algorithm ID.	

⌋(*RS_CRYPTO_02308*)

8.16.1.1.3.2 virtual ara::core::Result<ara::core::String> ara::crypto::cryp::CryptoProvider::ConvertToAlgName (AlgId *algId*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20712]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::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	
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.	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value	

⌋(*RS_CRYPTO_02308*)

8.16.1.1.3.3 virtual ara::core::Result<CryptoPrimitiveId::Category> ara::crypto::cryp::CryptoProvider::GetPrimitiveCategory (AlgId *algId*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20734]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::GetPrimitiveCategory(AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<CryptoPrimitiveId::Category> GetPrimitive Category (AlgId algId) const noexcept=0;`	
Parameters (in):	algId	the vendor specific binary algorithm ID
Return value:	ara::core::Result< CryptoPrimitive Id::Category >	the crypto primitive category of the crypto algorithm
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Get Crypto Primitive Category of specified algorithm.	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value	

⌋*(RS_CRYPTO_02308, RS_CRYPTO_02309)*

8.16.1.1.3.4 virtual ara::core::Result<void> ara::crypto::cryp::CryptoProvider::ReserveContexts (const ContextReservationMap & *reservationMap*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20713]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ReserveContexts(const ContextReservationMap &reservationMap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ReserveContexts (const Context ReservationMap &reservationMap) noexcept=0;`	
Parameters (in):	reservationMap	the contexts reservation map that defines minimal required size for each reserved Context slot
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Reserve memory for simultaneous hosting of all Contexts specified by the map.	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if reservationMap includes unknown algorithm identifiers [Error]: SecurityErrorDomain::kBadAlloc if the requested reservation cannot be executed	

⌋*(RS_CRYPTO_02404)*

8.16.1.1.3.5 virtual ara::core::Result<void> ara::crypto::cryp::CryptoProvider::ReserveObjects (const ObjectReservationMap & *reservationMap*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20714]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ReserveObjects(const ObjectReservationMap &reservationMap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ReserveObjects (const ObjectReservationMap &reservationMap) noexcept=0;`	
Parameters (in):	reservationMap	the objects reservation map that defines minimal required size for each reserved Object slot
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Reserve memory for simultaneous hosting of all Objects specified by the map.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if unknown or unsupported combination of object type and algorithm ID presents in the reservationMap [Error]: SecurityErrorDomain::kBadAlloc if the requested reservation cannot be executed	

⌋*(RS_CRYPTO_02404)*

8.16.1.1.3.6 virtual ara::core::Result<void> ara::crypto::cryp::CryptoProvider::ReserveContexts (std::size_t *quantity*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20715]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ReserveContexts(std::size_t quantity)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ReserveContexts (std::size_t quantity) noexcept=0;`	
Parameters (in):	quantity	the number of Centexts for reservation
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Reserve memory for simultaneous hosting of specified quantity of any type Contexts, i.e. maximal capacity will be reserved for each Context.	
Notes:	[Error]: SecurityErrorDomain::kBadAlloc if the requested reservation cannot be executed	

⌋*(RS_CRYPTO_02404)*

8.16.1.1.3.7 virtual ara::core::Result<void> ara::crypto::cryp::CryptoProvider::ReserveObjects (std::size_t *quantity*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20716]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ReserveObjects(std::size_t quantity)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ReserveObjects (std::size_t quantity) noexcept=0;`	
Parameters (in):	quantity	the number of Objects for reservation
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Reserve memory for simultaneous hosting of specified quantity of any type Objects, i.e. maximal capacity will be reserved for each Object.	
Notes:	[Error]: SecurityErrorDomain::kBadAlloc if the requested reservation cannot be executed	

⌋*(RS_CRYPTO_02404)*

8.16.1.1.3.8 virtual void ara::crypto::cryp::CryptoProvider::EnterRealTimeMode ( ) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20717]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoProvider::EnterRealTimeMode()
Scope:	class ara::crypto::cryp::CryptoProvider
Syntax:	`virtual void EnterRealTimeMode () noexcept=0;`
Return value:	None
Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Switches the memory management emgine of the current execution thread to the "Real-Time" mode.
Notes:	After entering to the "Real-Time" mode, any allocations of Contexts or Objects on the heap are prohibited. In the "Real-Time" mode only reserved Objects and Contexts can be used (see Reserve Contexts() and ReserveObjects() methods). Indexes of reserved Objects and Contexts slots are presented in the API by the types Reserved ObjectIndex and ReservedContextIndex respectively.

⌋*(RS_CRYPTO_02406)*

8.16.1.1.3.9 virtual void ara::crypto::cryp::CryptoProvider::LeaveRealTimeMode ( ) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20718]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoProvider::LeaveRealTimeMode()
Scope:	class ara::crypto::cryp::CryptoProvider
Syntax:	`virtual void LeaveRealTimeMode () noexcept=0;`
Return value:	None
Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Switche the memory management emgine of the current execution thread to the "Non-Real-Time" mode.
Notes:	After leaving the "Real-Time" mode, any allocations of Contexts or Objects on the heap are allowed. In the "Non-Real-Time" mode the reserved Objects and Contexts can be used too (see ReserveContexts() and ReserveObjects() methods). Indexes of reserved Objects and Contexts slots are presented in the API by the types Reserved ObjectIndex and ReservedContextIndex respectively.

⌋(*RS_CRYPTO_02406*)

8.16.1.1.3.10 virtual ara::core::Result<DomainParameters::Sptr> ara::crypto::cryp::CryptoProvider::AllocDomainParameters (AlgId *algId,* bool *isSession* = `false`, bool *isExportable* = `false`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20719]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::AllocDomainParameters(AlgId algId, bool isSession=false, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<DomainParameters::Sptr> AllocDomain Parameters (AlgId algId, bool isSession=false, bool is Exportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOn Heap) noexcept=0;`	
Parameters (in):	algId	an identifier of the algorithm for which the domain parameters are intended
	isSession	the "session" (or "temporary") attribute of the target domain parameters (if true)
	isExportable	the exportability attribute of the target domain parameters object
	reservedIndex	the optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Domain Parameters::Sptr >	shared smart pointer to the allocated domain parameter object
Exception Safety:	noexcept	

▽

△

Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Allocate an empty object of domain parameters for specified algorithm.
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if algId has an incorrect value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02108, RS_CRYPTO_02113, RS_CRYPTO_02404)*

8.16.1.1.3.11 virtual ara::core::Result<DomainParameters::Sptrc> ara::crypto::cryp::CryptoProvider::KnownDomainParameters (ara::core::StringView *oidName*, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20720]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::KnownDomainParameters(ara::core::StringView oidName, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<DomainParameters::Sptrc> KnownDomain Parameters (ara::core::StringView oidName, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0;`	
Parameters (in):	oidName	OID/Name of required domain parameters (names are case-insensitive)
	reservedIndex	the optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Domain Parameters::Sptrc >	shared smart pointer to the allocated domain parameter object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load a known domain parameters by their OID/Name.	
Notes:	Crypto Provider can share a single instance of named (i.e. constant) domain parameters between a few consumers. [Error]: SecurityErrorDomain::kUnknownIdentifier if the oidName argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02108, RS_CRYPTO_02404)*

8.16.1.1.3.12 virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::CryptoProvider::GenerateSymmetricKey (AlgId *algId*, Key::Usage *allowedUsage*, bool *isSession* = `true`, bool *isExportable* = `false`, DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20721]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::GenerateSymmetricKey(AlgId algId, Key::Usage allowed Usage, bool isSession=true, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> GenerateSymmetricKey (AlgId algId, Key::Usage allowedUsage, bool isSession=true, bool is Exportable=false, DomainParameters::Sptrc params=nullptr, Reserved ObjectIndex reservedIndex=kAllocObjectOnHeap) 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 KeyMaterial)
	isSession	the "session" (or "temporary") attribute of the target key (if true)
	isExportable	the exportability attribute of the target key (if true)
	params	the optional pointer to Domain Parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	the optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Symmetric Key::Uptrc >	smart unique pointer to the created symmetric key object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
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.
Notes:	If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the generated key object. 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). [Error]: SecurityErrorDomain::kUnknownIdentifier if algId has an unsupported value [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kIncompatibleArguments if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)

⌋*(RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02113, RS_CRYPTO_02115, RS_CRYPTO_02404)*

8.16.1.1.3.13 virtual ara::core::Result<PrivateKey::Uptrc> ara::crypto::cryp::CryptoProvider::GeneratePrivateKey (AlgId *algId,* Key::Usage *allowedUsage,* bool *isSession* = `false`, bool *isExportable* = `false`, DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20722]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoProvider::GeneratePrivateKey(AlgId algId, Key::Usage allowedUsage, bool isSession=false, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)
Scope:	class ara::crypto::cryp::CryptoProvider

▽

△

Syntax:	`virtual ara::core::Result<PrivateKey::Uptrc> GeneratePrivateKey (AlgId algId, Key::Usage allowedUsage, bool isSession=false, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) 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 KeyMaterial)
	isSession	the "session" (or "temporary") attribute for the target key (if true)
	isExportable	the exportability attribute of the target key (if true)
	params	the optional pointer to Domain Parameters required for full specification of the transformation
	reservedIndex	An optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< PrivateKey::Uptrc >	smart unique pointer to the created private key object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
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.	
Notes:	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! If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! [Error]: SecurityErrorDomain::kUnknownIdentifier if algId has an unsupported value [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kIncompatibleArguments if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)	

⌋*(RS_CRYPTO_02003, RS_CRYPTO_02101, RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02113, RS_CRYPTO_02115, RS_CRYPTO_02404)*

8.16.1.1.3.14 virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::CryptoProvider::GenerateSeed (AlgId *algId*, SecretSeed::Usage *allowedUsage*, bool *isSession* = `true`, bool *isExportable* = `false`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20723]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::GenerateSeed(AlgId algId, SecretSeed::Usage allowed Usage, bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=k AllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> GenerateSeed (AlgId alg Id, SecretSeed::Usage allowedUsage, bool isSession=true, bool is Exportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOn Heap) 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 KeyMaterial)
	isSession	the "session" (or "temporary") attribute of the target seed (if true)
	isExportable	the exportability attribute of the target seed (if true)
	reservedIndex	the optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to generated SecretSeed object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Generate a random Secret Seed object of requested algorithm.	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if algId has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kIncompatibleArguments if allowedUsage argument is incompatible with target algorithm algId (note: it is an optional error condition for this method)	

⌋*(RS_CRYPTO_02007, RS_CRYPTO_02404)*

8.16.1.1.3.15 virtual ara::core::Result<std::size_t> ara::crypto::cryp::CryptoProvider::GetSerializedSize (ObjectType *objectType,* AlgId *algId,* Serializable::FormatId *formatId* = Serializable::kFormatDefault) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20724]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::GetSerializedSize(ObjectType objectType, AlgId algId, Serializable::FormatId formatId=Serializable::kFormatDefault)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<std::size_t> GetSerializedSize (ObjectType objectType, AlgId algId, Serializable::FormatId formatId=Serializable::kFormatDefault) const noexcept=0;`	
Parameters (in):	objectType	the type of the target object
	algId	the Crypto Provider algorithm ID of the target object
	formatId	the Crypto Provider specific identifier of the output format
Return value:	ara::core::Result< std::size_t >	size required for storing of the object serialized in the specified format
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Return required buffer size for serialization of an object in specific format.	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if any argument has an unsupported value [Error]: SecurityErrorDomain::kIncompatibleArguments if any pair of the arguments are incompatible	

⌋*(RS_CRYPTO_02404)*

8.16.1.1.3.16 virtual ara::core::Result<std::size_t> ara::crypto::cryp::CryptoProvider::GetStorageSize (ObjectType *objectType,* AlgId *algId*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20725]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::GetStorageSize(ObjectType objectType, AlgId algId)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<std::size_t> GetStorageSize (ObjectType objectType, AlgId algId) const noexcept=0;`	
Parameters (in):	objectType	the type of the target object
	algId	a Crypto Provider algorithm ID of the target object
Return value:	ara::core::Result< std::size_t >	size required for storing of the object in the Key Storage

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Return required capacity of a key slot for saving of the object.
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if any argument has an unsupported value [Error]: SecurityErrorDomain::kIncompatibleArguments if the arguments are incompatible

⌋*(RS_CRYPTO_02404)*

8.16.1.1.3.17 virtual ara::core::Result<TrustedContainer::Uptr> ara::crypto::cryp::CryptoProvider::AllocVolatileContainer (std::size_t *capacity = 0*) `[pure virtual], [noexcept]`

[SWS_CRYPT_20726]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::AllocVolatileContainer(std::size_t capacity=0)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<TrustedContainer::Uptr> AllocVolatile Container (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< Trusted Container::Uptr >	unique smart pointer to an allocated volatile trusted container
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Allocate a Volatile (virtual) Trusted Container according to directly specified capacity.	
Notes:	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. [Error]: SecurityErrorDomain::kBadAlloc if the requested allocation cannot be executed	

⌋*(RS_CRYPTO_02404)*

8.16.1.1.3.18 virtual ara::core::Result<TrustedContainer::Uptr> ara::crypto::cryp::CryptoProvider::AllocVolatileContainer (const ContainedObjectsList & *objectsList*) `[pure virtual], [noexcept]`

[SWS_CRYPT_20727]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::AllocVolatileContainer(const ContainedObjectsList &objects List)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<TrustedContainer::Uptr> AllocVolatile Container (const ContainedObjectsList &objectsList) noexcept=0;`	
Parameters (in):	objectsList	the list of objects that can be stored to this volatile trusted container
Return value:	ara::core::Result< Trusted Container::Uptr >	unique smart pointer to an allocated volatile trusted container
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Allocate a Volatile (virtual) Trusted Container according to indirect specification of a minimal required capacity for hosting of any listed object.	
Notes:	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. [Error]: SecurityErrorDomain::kInvalidArgument if unsupported combination of object type and algorithm ID presents in the list [Error]: SecurityErrorDomain::kBadAlloc if the requested allocation cannot be executed	

⌋(*RS_CRYPTO_02404*)

8.16.1.1.3.19 virtual ara::core::Result<std::size_t> ara::crypto::cryp::CryptoProvider::ExportSecuredObject (const CryptoObject & *object*, SymmetricKeyWrapperCtx & *transportContext*, WritableMemRegion *serialized* = WritableMemRegion*()*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20728]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ExportSecuredObject(const CryptoObject &object, SymmetricKeyWrapperCtx &transportContext, WritableMemRegion serialized=WritableMem Region())	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<std::size_t> ExportSecuredObject (const CryptoObject &object, SymmetricKeyWrapperCtx &transportContext, WritableMemRegion serialized=WritableMemRegion()) noexcept=0;`	
Parameters (in):	object	the crypto object for export
	transportContext	the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyExporting)
Parameters (out):	serialized	the output buffer for the serialized object
Return value:	ara::core::Result< std::size_t >	actual capacity required for the serialized data
Exception Safety:	noexcept	

▽

△

Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Export a crypto object in a secure manner.
Notes:	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 GUID) can be exported! [Error]: SecurityErrorDomain::kInsufficientCapacity if size of the serialized buffer is not enough for saving the output data [Error]: SecurityErrorDomain::kIncompleteArgState if the transportContext is not initialized [Error]: SecurityErrorDomain::kIncompatibleObject if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

8.16.1.1.3.20 virtual ara::core::Result<std::size_t> ara::crypto::cryp::CryptoProvider::ExportSecuredObject (const TrustedContainer & *container*, SymmetricKeyWrapperCtx & *transportContext*, WritableMemRegion *serialized* = WritableMemRegion *()*) `[pure virtual], [noexcept]`

[SWS_CRYPT_20729]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ExportSecuredObject(const TrustedContainer &container, SymmetricKeyWrapperCtx &transportContext, WritableMemRegion serialized=WritableMem Region()	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<std::size_t> ExportSecuredObject (const TrustedContainer &container, SymmetricKeyWrapperCtx &transportContext, WritableMemRegion serialized=WritableMemRegion()) noexcept=0;`	
Parameters (in):	container	the trusted container that contains an object for export
	transportContext	the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyExporting)
Parameters (out):	serialized	the output buffer for the serialized object
Return value:	ara::core::Result< std::size_t >	actual capacity required for the serialized data
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Export securely an object directly from a trusted container (i.e. without an intermediate creation of a crypto object).	

▽

△

Notes:	if (serialized == nullptr) then the method returns required size only, but content of the transport Context stays unchanged. This method can be used for re-exporting of just imported object but on another transport key. [Error]: SecurityErrorDomain::kEmptyContainer if the container is empty [Error]: SecurityErrorDomain::kInsufficientCapacity if size of the serialized buffer is not enough for saving the output data [Error]: SecurityErrorDomain::kIncompleteArgState if the transportContext is not initialized [Error]: SecurityErrorDomain::kIncompatibleObject if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method)

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

8.16.1.1.3.21 virtual ara::core::Result<void> ara::crypto::cryp::CryptoProvider::ImportSecuredObject (TrustedContainer & *container*, ReadOnlyMemRegion *serialized*, SymmetricKeyWrapperCtx & *transportContext*, bool *isExportable* = `false`, ObjectType *expectedObject* = `ObjectType::kUnknown`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20730]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ImportSecuredObject(TrustedContainer &container, ReadOnlyMemRegion serialized, SymmetricKeyWrapperCtx &transportContext, bool isExportable=false, ObjectType expectedObject=ObjectType::kUnknown)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ImportSecuredObject (TrustedContainer &container, ReadOnlyMemRegion serialized, SymmetricKeyWrapperCtx &transportContext, bool isExportable=false, ObjectType expectedObject=ObjectType::kUnknown) noexcept=0;`	
Parameters (in):	serialized	the memory region that contains a securely serialized object that should be imported to the trusted container
	transportContext	the symmetric key wrap context initialized by a transport key (allowed usage: kAllowKeyImporting)
	isExportable	the exportability attribute of the target object (this value for public keys and public domain parameters should be ignored, because they are always exportable)
	expectedObject	the expected object type (default value ObjectType::kUnknown means without check)
Parameters (out):	container	the prealocated volatile trusted container for storing of the imported object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Import securely serialized object to a temporary (volatile) trusted container for following processing (without allocation of a crypto object context).
Notes:	[Error]: SecurityErrorDomain::kUnexpectedValue if the serialized contains incorrect data [Error]: SecurityErrorDomain::kBadObjectType if (expectedObject != ObjectType::kUnknown), but the actual object type differs from the expected one [Error]: SecurityErrorDomain::kIncompleteArgState if the transportContext is not initialized [Error]: SecurityErrorDomain::kIncompatibleObject if a key loaded to the transportContext doesn't have required attributes (note: it is an optional error condition for this method) [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the container is not enough to save the deserialized object

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112, RS_CRYPTO_02113)*

8.16.1.1.3.22 virtual ara::core::Result<std::size_t> ara::crypto::cryp::CryptoProvider::ExportPublicObject (const TrustedContainer & *container*, WritableMemRegion *serialized* = WritableMemRegion (), Serializable::FormatId *formatId* = Serializable::kFormatDefault) [pure virtual], [noexcept]

[SWS_CRYPT_20731]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ExportPublicObject(const TrustedContainer &container, WritableMemRegion serialized=WritableMemRegion()	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<std::size_t> ExportPublicObject (const TrustedContainer &container, WritableMemRegion serialized=WritableMemRegion(), Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0;`	
Parameters (in):	container	the trusted container that contains an object for export
	formatId	the Crypto Provider specific identifier of the output format
Parameters (out):	serialized	the output buffer for the serialized object
Return value:	ara::core::Result< std::size_t >	actual capacity required for the serialized data
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Export publicly an object from a trusted container (i.e. without an intermediate creation of a crypto object context).	
Notes:	[Error]: SecurityErrorDomain::kEmptyContainer if the container is empty [Error]: SecurityErrorDomain::kUnexpectedValue if the container contains a secret crypto object [Error]: SecurityErrorDomain::kInsufficientCapacity if (serialized.empty() == false), but its capacity is not enough for storing result	

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

8.16.1.1.3.23 virtual ara::core::Result<void> ara::crypto::cryp::CryptoProvider::ImportPublicObject (TrustedContainer & *container*, ReadOnlyMemRegion *serialized*, ObjectType *expectedObject* = *ObjectType::kUnknown*) [pure virtual], [noexcept]

[SWS_CRYPT_20732]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::ImportPublicObject(TrustedContainer &container, ReadOnly MemRegion serialized, ObjectType expectedObject=ObjectType::kUnknown)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> ImportPublicObject (TrustedContainer &container, ReadOnlyMemRegion serialized, ObjectType expected Object=ObjectType::kUnknown) noexcept=0;`	
Parameters (in):	serialized	the memory region that contains a securely serialized object that should be imported to the trusted container
	expectedObject	the expected object type (default value Object Type::kUnknown means without check)
Parameters (out):	container	the prealocated volatile trusted container for storing of the imported object
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Import publicly serialized object to a temporary (volatile) trusted container for following processing (without allocation of a crypto object context).	
Notes:	If (expectedObject != ObjectType::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 [Error]: SecurityErrorDomain::kUnexpectedValue if the serialized contains incorrect data [Error]: SecurityErrorDomain::kBadObjectType if (expectedObject != ObjectType::kUnknown), but the actual object type differs from the expected one [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the container is not enough to save the deserialized object	

⌋*(RS_CRYPTO_02105, RS_CRYPTO_02112)*

8.16.1.1.3.24 virtual ara::core::Result<CryptoObject::Uptrc> ara::crypto::cryp::CryptoProvider::LoadObject (const TrustedContainer & *container*, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) [pure virtual], [noexcept]

[SWS_CRYPT_20733]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::LoadObject(const TrustedContainer &container, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<CryptoObject::Uptrc> LoadObject (const TrustedContainer &container, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0;`	
Parameters (in):	container	the trusted container that contains the crypto object for loading
	reservedIndex	the optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< CryptoObject::Uptrc >	unique smart pointer to the created object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load any crypto object from a "trusted container".	
Notes:	[Error]: SecurityErrorDomain::kEmptyContainer if the container is empty [Error]: SecurityErrorDomain::kUnexpectedValue if the container content is damaged [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reservedIndex is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible This method is one of the "binding" methods between a Crypto Provider and the Key Storage Provider. Also this method may implement the Policy Enforcement Point (PEP) for access control via Identity and Access Management (IAM).	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002, RS_CRYPTO_02404)*

8.16.1.1.3.25 template<typename ExpectedObject > ara::core::Result<typename ExpectedObject::Uptrc> ara::crypto::cryp::CryptoProvider::LoadConcreteObject (const TrustedContainer & *container*, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[noexcept]`

[SWS_CRYPT_20760]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoProvider::LoadConcreteObject(const TrustedContainer &container, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)
Scope:	class ara::crypto::cryp::CryptoProvider

▽

△

Syntax:	`template <typename ExpectedObject> inline ara::core::Result<typename ExpectedObject::Uptrc> LoadConcreteObject (const TrustedContainer &container, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept;`	
Template param:	ExpectedObject	the expected type of concrete object
Parameters (in):	container	the trusted container that contains the crypto object for loading
	reservedIndex	the optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< typename ExpectedObject::Uptrc >	unique smart pointer to the created object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Load concrete crypto object (of specified type) from a "trusted container".	
Notes:	[Error]: SecurityErrorDomain::kEmptyContainer if the container is empty [Error]: SecurityErrorDomain::kUnexpectedValue if the container content is damaged [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reservedIndex is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kBadObjectType if an actual type of the container content differs from the expected type of concrete object	

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02002, RS_CRYPTO_02404)*

8.16.1.1.3.26 virtual ara::core::Result<PasswordCache::Uptr> ara::crypto::cryp::CryptoProvider::AllocPasswordCache (std::size_t *maximalLength,* std::size_t *requiredLength,* unsigned *requiredComplexity,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20736]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::CryptoProvider::AllocPasswordCache(std::size_t maximalLength, std::size_t requiredLength, unsigned requiredComplexity, ReservedContextIndex reservedIndex=kAllocContextOnHeap)
Scope:	class ara::crypto::cryp::CryptoProvider

▽

△

Syntax:	`virtual ara::core::Result<PasswordCache::Uptr> AllocPasswordCache (std::size_t maximalLength, std::size_t requiredLength, unsigned requiredComplexity, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	maximalLength	the maximal supported length of a target password (in characters)
	requiredLength	the minimal required length of a target password (in characters)
	requiredComplexity	the minimal required complexity of a target password (0 means "no requirements")
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< PasswordCache::Uptr >	smart unique pointer to the allocated password context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Allocate new Password Cache context.	
Notes:	The complexity is measured by a number symbols' categories (e.g.: lower/upper cases, numbers, special symbols). A maximal supported value of the argument maximalLength can be restricted by an implementation. [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reservedIndex is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kInvalidArgument if any of arguments has unsupported value	

⌋*(RS_CRYPTO_02106, RS_CRYPTO_02404)*

8.16.1.1.3.27 virtual ara::core::Result<PasswordHash::Uptr> ara::crypto::cryp::CryptoProvider::HashPassword (HashFunctionCtx & *hashCtx*, const PasswordCache & *password*, bool *isSession* `= false`, bool *isExportable* `= false`, ReservedObjectIndex *reservedIndex* `=` kAllocObjectOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20738]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::HashPassword(HashFunctionCtx &hashCtx, const PasswordCache &password, bool isSession=false, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<PasswordHash::Uptr> HashPassword (HashFunctionCtx &hashCtx, const PasswordCache &password, bool isSession=false, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept=0;`	
Parameters (in):	hashCtx	the hash-function context that should be used in this context
	password	the original password that should be hashed
	isSession	the "session" (or "temporary") attribute for the password hash object (if true)
	isExportable	the exportability attribute for the password hash object (if true)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< PasswordHash::Uptr >	unique smart pointer to the allocated password hash context or nullptr if the container doesn't have an object of this type
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a password hash object.	
Notes:	An internal hash-function context required by the password hash context should be preallocated by this method too. Any serializable (i.e. savable/non-session or exportable) password hash object must generate own COUID! [Error]: SecurityErrorDomain::kIncompleteArgState if the configuring of the hash-function context (i.e. hashCtx) is not finished yet (e.g. domain parameters are required, but not set yet) [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reservedIndex is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02106, RS_CRYPTO_02113, RS_CRYPTO_02404)*

8.16.1.1.3.28 virtual RandomGeneratorCtx::Sptr ara::crypto::cryp::CryptoProvider::DefaultRng ( ) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20739]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::DefaultRng()	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual RandomGeneratorCtx::Sptr DefaultRng () noexcept=0;`	
Return value:	RandomGeneratorCtx::Sptr	shared smart pointer to default Random Number Generator
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Return a shared pointer to instance of default Random Number Generator (RNG) used by the Crypto Provider internaly.	
Notes:	The default RNG should be the best one (i.e. a most secure) from all supported RNG (ideally TRNG).	

⌋*(RS_CRYPTO_02206, RS_CRYPTO_02404)*

8.16.1.1.3.29 virtual ara::core::Result<void> ara::crypto::cryp::CryptoProvider::SetDefaultRng (RandomGeneratorCtx::Sptr *rng* = *nullptr*) [pure virtual], [noexcept]

[SWS_CRYPT_20740]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::SetDefaultRng(RandomGeneratorCtx::Sptr rng=nullptr)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<void> SetDefaultRng (RandomGeneratorCtx::Sptr rng=nullptr) noexcept=0;`	
Parameters (in):	rng	the shared smart pointer to an instance of the RNG context
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Set default Random Number Generator (RNG) instance.	
Notes:	If (rng == nullptr) then internal pointer of default RNG should be returned to original Crypto Proider-specific default RNG instance. Reconfiguration executed by this method affects on the Crypto Provider instance in current process only! [Error]: SecurityErrorDomain::kIncompleteArgState if (rng != nullptr), but provided RNG instance is not initialized yet	

⌋*(RS_CRYPTO_02206, RS_CRYPTO_02404)*

8.16.1.1.3.30 virtual ara::core::Result<RandomGeneratorCtx::Sptr> ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx (AlgId *algId,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20741]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateRandomGeneratorCtx(AlgId algId, ReservedContext Index reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<RandomGeneratorCtx::Sptr> CreateRandom GeneratorCtx (AlgId algId, ReservedContextIndex reservedIndex=kAlloc ContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of target RNG algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< RandomGenerator Ctx::Sptr >	shared smart pointer to the created RNG context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a Random Number Generator (RNG) context.	
Notes:	A fully Deterministic RNG should be used only for debugging purposes, but any RNG used "in the field" should support an internal entropy source (not controllable by application). [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02206, RS_CRYPTO_02404)*

8.16.1.1.3.31 virtual ara::core::Result<SymmetricBlockCipherCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSymmetricBlockCipherCtx (AlgId *algId,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20742]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateSymmetricBlockCipherCtx(AlgId algId, Reserved ContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SymmetricBlockCipherCtx::Uptr> Create SymmetricBlockCipherCtx (AlgId algId, ReservedContextIndex reserved Index=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< SymmetricBlock CipherCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric block cipher context.	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02201, RS_CRYPTO_02404)*

8.16.1.1.3.32 virtual ara::core::Result<SymmetricKeyWrapperCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSymmetricKeyWrapperCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20743]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateSymmetricKeyWrapperCtx(AlgId algId, Reserved ContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SymmetricKeyWrapperCtx::Uptr> Create SymmetricKeyWrapperCtx (AlgId algId, ReservedContextIndex reserved Index=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap

▽

△

Return value:	ara::core::Result< SymmetricKey WrapperCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric key-wrap algorithm context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from symmetric key-wrapping [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02104, RS_CRYPTO_02208, RS_CRYPTO_02404)*

8.16.1.1.3.33 virtual ara::core::Result<StreamCipherCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateStreamCipherCtx (AlgId *algId,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20744]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateStreamCipherCtx(AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<StreamCipherCtx::Uptr> CreateStreamCipherCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< StreamCipher Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric stream cipher context.	

▽

△

Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from symmetric stream cipher [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02201, RS_CRYPTO_02404)*

8.16.1.1.3.34 virtual ara::core::Result<AuthnStreamCipherCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateAuthnStreamCipherCtx (AlgId *algId,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20745]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateAuthnStreamCipherCtx(AlgId algId, ReservedContext Index reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<AuthnStreamCipherCtx::Uptr> CreateAuthn StreamCipherCtx (AlgId algId, ReservedContextIndex reservedIndex=k AllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< AuthnStreamCipher Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric authenticated stream cipher context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from symmetric authenticated stream cipher [Error]: SecurityErrorDomain::kInvalidArgument [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋(*RS_CRYPTO_02207*, *RS_CRYPTO_02404*)

8.16.1.1.3.35 virtual ara::core::Result<MessageAuthnCodeCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateMessageAuthnCodeCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20746]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateMessageAuthnCodeCtx(AlgId algId, ReservedContext Index reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<MessageAuthnCodeCtx::Uptr> CreateMessage AuthnCodeCtx (AlgId algId, ReservedContextIndex reservedIndex=kAlloc ContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< MessageAuthnCode Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric message authentication code context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from symmetric message authentication code [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋(*RS_CRYPTO_02203*, *RS_CRYPTO_02404*)

8.16.1.1.3.36 virtual ara::core::Result<HashFunctionCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateHashFunctionCtx (AlgId *algId,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) [pure virtual], [noexcept]

[SWS_CRYPT_20747]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateHashFunctionCtx(AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<HashFunctionCtx::Uptr> CreateHashFunctionCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< HashFunction Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a hash function context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from hash function [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02205, RS_CRYPTO_02404)*

8.16.1.1.3.37 virtual ara::core::Result<KeyDerivationFunctionCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateKeyDerivationFunctionCtx (AlgId *algId,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) [pure virtual], [noexcept]

[SWS_CRYPT_20748]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateKeyDerivationFunctionCtx(AlgId algId, Reserved ContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<KeyDerivationFunctionCtx::Uptr> CreateKey DerivationFunctionCtx (AlgId algId, ReservedContextIndex reserved Index=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< KeyDerivation FunctionCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a key derivation function context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from key derivation function [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02103, RS_CRYPTO_02404)*

8.16.1.1.3.38 virtual ara::core::Result<KeyDiversifierCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateKeyDiversifierCtx (AlgId *masterAlgId*, std::size_t *slaveKeyLength*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20749]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateKeyDiversifierCtx(AlgId masterAlgId, std::size_t slave KeyLength, ReservedContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<KeyDiversifierCtx::Uptr> CreateKey DiversifierCtx (AlgId masterAlgId, std::size_t slaveKeyLength, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	masterAlgId	the crypto algorithm identifier of master-keys

▽

△

	slaveKeyLength	the length of target slave-keys (derived from the master) in bits
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< KeyDiversifier Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a symmetric key diversification context.	
Notes:	slaveAlgId can have partial specification (only algorithm family and key length are required, but the mode and padding are optional). [Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from symmetric key diversification [Error]: SecurityErrorDomain::kIncompatibleArguments if requested slaveKeyLength value is unsupported for specified masterAlgId [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02103, RS_CRYPTO_02404)*

8.16.1.1.3.39 virtual ara::core::Result<EncryptorPublicCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateEncryptorPublicCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20750]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateEncryptorPublicCtx(AlgId algId, ReservedContext Index reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<EncryptorPublicCtx::Uptr> CreateEncryptor PublicCtx (AlgId algId, ReservedContextIndex reservedIndex=kAlloc ContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric encryption/decryption algorithm

▽

△

	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< EncryptorPublic Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create an encryption public key context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02404)*

8.16.1.1.3.40 virtual ara::core::Result<DecryptorPrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateDecryptorPrivateCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20751]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateDecryptorPrivateCtx(AlgId algId, ReservedContext Index reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<DecryptorPrivateCtx::Uptr> CreateDecryptor PrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAlloc ContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric encryption/decryption algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< DecryptorPrivate Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

Header file:	#include "ara/crypto/cryp/crypto_provider.h"
Description:	Create a decryption private key context.
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from asymmetric encryption/decryption [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02404)*

8.16.1.1.3.41 virtual ara::core::Result<KeyEncapsulatorPublicCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateKeyEncapsulatorPublicCtx (AlgId *algId,* ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20752]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateKeyEncapsulatorPublicCtx(AlgId algId, Reserved ContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<KeyEncapsulatorPublicCtx::Uptr> CreateKey EncapsulatorPublicCtx (AlgId algId, ReservedContextIndex reserved Index=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target KEM crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< KeyEncapsulator PublicCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a key-encapsulator public key context of a Key Encapsulation Mechanism (KEM).	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from asymmetric KEM [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋(*RS_CRYPTO_02104*, *RS_CRYPTO_02209*, *RS_CRYPTO_02404*)

8.16.1.1.3.42 virtual ara::core::Result<KeyDecapsulatorPrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateKeyDecapsulatorPrivateCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20753]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateKeyDecapsulatorPrivateCtx(AlgId algId, Reserved ContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<KeyDecapsulatorPrivateCtx::Uptr> CreateKey DecapsulatorPrivateCtx (AlgId algId, ReservedContextIndex reserved Index=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target KEM crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< KeyDecapsulator PrivateCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a key-decapsulator private key context of a Key Encapsulation Mechanism (KEM).	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from asymmetric KEM [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋(*RS_CRYPTO_02104*, *RS_CRYPTO_02209*, *RS_CRYPTO_02404*)

8.16.1.1.3.43 virtual ara::core::Result<SigEncodePrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSigEncodePrivateCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) [pure virtual], [noexcept]

[SWS_CRYPT_20754]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateSigEncodePrivateCtx(AlgId algId, ReservedContext Index reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SigEncodePrivateCtx::Uptr> CreateSigEncode PrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAlloc ContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< SigEncodePrivate Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a signature encoding private key context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204, RS_CRYPTO_02404)*

8.16.1.1.3.44 virtual ara::core::Result<MsgRecoveryPublicCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateMsgRecoveryPublicCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) [pure virtual], [noexcept]

[SWS_CRYPT_20755]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateMsgRecoveryPublicCtx(AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<MsgRecoveryPublicCtx::Uptr> CreateMsgRecoveryPublicCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target asymmetric crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< MsgRecoveryPublicCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a message recovery public key context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from asymmetric signature encoding with message recovery [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reservedIndex is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02202, RS_CRYPTO_02204, RS_CRYPTO_02404)*

8.16.1.1.3.45 virtual ara::core::Result<SignerPrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateSignerPrivateCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual], [noexcept]`

[SWS_CRYPT_20756]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateSignerPrivateCtx(AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<SignerPrivateCtx::Uptr> CreateSignerPrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target signature crypto algorithm

▽

△

	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< SignerPrivate Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a signature private key context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from private key signature [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02404)*

8.16.1.1.3.46 virtual ara::core::Result<VerifierPublicCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateVerifierPublicCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20757]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateVerifierPublicCtx(AlgId algId, ReservedContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<VerifierPublicCtx::Uptr> CreateVerifier PublicCtx (AlgId algId, ReservedContextIndex reservedIndex=kAlloc ContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target signature crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< VerifierPublic Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	

▽

△

Description:	Create a signature verification public key context.
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from public key signature verification [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02204, RS_CRYPTO_02404)*

8.16.1.1.3.47 virtual ara::core::Result<KeyAgreementPrivateCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateKeyAgreementPrivateCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20758]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateKeyAgreementPrivateCtx(AlgId algId, Reserved ContextIndex reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<KeyAgreementPrivateCtx::Uptr> CreateKey AgreementPrivateCtx (AlgId algId, ReservedContextIndex reservedIndex=k AllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target key-agreement crypto algorithm
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< KeyAgreement PrivateCtx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create a key-agreement private key context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from key-agreement [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋(*RS_CRYPTO_02104*, *RS_CRYPTO_02404*)

8.16.1.1.3.48 virtual ara::core::Result<X509RequestSignerCtx::Uptr> ara::crypto::cryp::CryptoProvider::CreateX509RequestSignerCtx (AlgId *algId*, ReservedContextIndex *reservedIndex* = kAllocContextOnHeap) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20759]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::CryptoProvider::CreateX509RequestSignerCtx(AlgId algId, ReservedContext Index reservedIndex=kAllocContextOnHeap)	
Scope:	class ara::crypto::cryp::CryptoProvider	
Syntax:	`virtual ara::core::Result<X509RequestSignerCtx::Uptr> Create X509RequestSignerCtx (AlgId algId, ReservedContextIndex reserved Index=kAllocContextOnHeap) noexcept=0;`	
Parameters (in):	algId	identifier of the target signature crypto algorithm that should be used for hashing and signature of certification requests produced by it
	reservedIndex	an optional index of reserved Context slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< X509RequestSigner Ctx::Uptr >	unique smart pointer to the created context
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/crypto_provider.h"	
Description:	Create an X.509 certificate request signer context.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if algId argument specifies a crypto algorithm different from private key signature or doesn't include hash algorithm specification [Error]: SecurityErrorDomain::kUnknownIdentifier if algId argument has an unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reserved Index is not enough for placing of the target context [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋(*RS_CRYPTO_02306*, *RS_CRYPTO_02307*, *RS_CRYPTO_02404*)

8.17 Top-level object interfaces

Classes

- class ara::crypto::cryp::DomainParameters
- class ara::crypto::cryp::PasswordHash
- class ara::crypto::cryp::PrivateKey
- class ara::crypto::cryp::PublicKey
- class ara::crypto::cryp::SecretSeed
- class ara::crypto::cryp::Signature
- class ara::crypto::cryp::SymmetricKey
- class ara::crypto::cryp::X509CertRequest
- class ara::crypto::cryp::X509PublicKeyInfo
- class ara::crypto::cryp::X509Signature

Detailed Description

This group consists of top-level interfaces of cryptographic objects available for manupulation by consumer applications.

8.17.1 Class Documentation

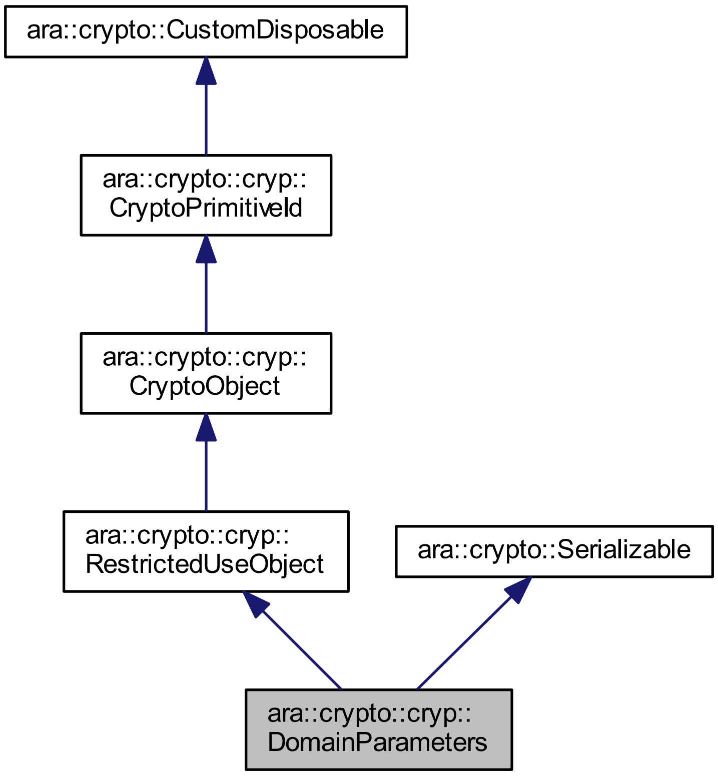
8.17.1.1 class ara::crypto::cryp::DomainParameters

[SWS_CRYPT_20900]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::DomainParameters
Scope:	namespace ara::crypto::cryp
Base class:	std::enable_shared_from_this< DomainParameters >
Syntax:	`class DomainParameters : public enable_shared_from_this< DomainParameters > {...};`
Header file:	#include "ara/crypto/cryp/domain_parameters.h"
Description:	Generic Domain Parameters interface.
Notes:	Any user of this interface should create shared pointers to it only by calls of the method shared_from_this()!

⌋*(RS_CRYPTO_02108)*

Inheritance diagram for ara::crypto::cryp::DomainParameters:



Public Types

- using Sptr = std::shared_ptr< DomainParameters >
- using Sptrc = std::shared_ptr< const DomainParameters >
- using Uptr = std::unique_ptr< DomainParameters, CustomDeleter >
- using Uptrc = std::unique_ptr< const DomainParameters, CustomDeleter >

Public Member Functions

- virtual bool IsSecret () const noexcept=0
- virtual std::size_t GetParametersCount () const noexcept=0
- virtual std::size_t GetMaxParameterNameLength () const noexcept=0

- virtual ara::core::Result< std::size_t > GetParameterName (std::size_t index, ara::core::String ∗name=nullptr) const noexcept=0
- virtual ara::core::Result< std::size_t > ExpectedParameterSize (std::size_t index) const noexcept=0
- virtual ara::core::Result< void > SetParameter (std::size_t index, ReadOnlyMemRegion value) noexcept=0
- virtual bool IsCompleted () const noexcept=0
- virtual bool Complete (Usage allowedUsage) noexcept=0
- virtual ara::core::StringView GetUniqueName () const noexcept=0

Additional Inherited Members

8.17.1.1.1 Member Typedef Documentation

8.17.1.1.1.1 using ara::crypto::cryp::DomainParameters::Sptr = std::shared_ptr<DomainParameters>

[SWS_CRYPT_20901]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::DomainParameters::Sptr
Scope:	class ara::crypto::cryp::DomainParameters
Derived from:	std::shared_ptr<DomainParameters>
Syntax:	`using ara::crypto::cryp::DomainParameters::Sptr = std::shared_ptr<DomainParameters>;`
Header file:	#include "ara/crypto/cryp/domain_parameters.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.17.1.1.1.2 using ara::crypto::cryp::DomainParameters::Sptrc = std::shared_ptr<const DomainParameters>

[SWS_CRYPT_20902]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::DomainParameters::Sptrc
Scope:	class ara::crypto::cryp::DomainParameters
Derived from:	std::shared_ptr<const DomainParameters>

▽

△

Syntax:	`using ara::crypto::cryp::DomainParameters::Sptrc = std::shared_ptr<const DomainParameters>;`
Header file:	#include "ara/crypto/cryp/domain_parameters.h"
Description:	Shared smart pointer of the constant interface.

⌋*(RS_CRYPTO_02311)*

8.17.1.1.1.3 using ara::crypto::cryp::DomainParameters::Uptr = std::unique_ptr<DomainParameters, CustomDeleter>

[SWS_CRYPT_20903]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::DomainParameters::Uptr
Scope:	class ara::crypto::cryp::DomainParameters
Derived from:	std::unique_ptr<DomainParameters, CustomDeleter>
Syntax:	`using ara::crypto::cryp::DomainParameters::Uptr = std::unique_ptr<DomainParameters, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/domain_parameters.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.17.1.1.1.4 using ara::crypto::cryp::DomainParameters::Uptrc = std::unique_ptr<const DomainParameters, CustomDeleter>

[SWS_CRYPT_20904]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::DomainParameters::Uptrc
Scope:	class ara::crypto::cryp::DomainParameters
Derived from:	std::unique_ptr<const DomainParameters, CustomDeleter>
Syntax:	`using ara::crypto::cryp::DomainParameters::Uptrc = std::unique_ptr<const DomainParameters, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/domain_parameters.h"
Description:	Unique smart pointer of the constant interface.

⌋*(RS_CRYPTO_02311)*

8.17.1.1.2 Member Function Documentation

8.17.1.1.2.1 virtual bool ara::crypto::cryp::DomainParameters::IsSecret ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20911]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::IsSecret()	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual bool IsSecret () const noexcept=0;`	
Return value:	bool	true if the set of these parameters is secret.
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Get secrecy attribute of these parameters set.	

⌋*(RS_CRYPTO_02309)*

8.17.1.1.2.2 virtual std::size_t ara::crypto::cryp::DomainParameters::GetParametersCount ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20912]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::GetParametersCount()	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual std::size_t GetParametersCount () const noexcept=0;`	
Return value:	std::size_t	number of supported parameters
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Get number of supported parameters.	

⌋*(RS_CRYPTO_02309)*

8.17.1.1.2.3 virtual std::size_t ara::crypto::cryp::DomainParameters::GetMaxParameterNameLength ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20913]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::GetMaxParameterNameLength()	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual std::size_t GetMaxParameterNameLength () const noexcept=0;`	
Return value:	std::size_t	maximal length between all parameters' names
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Get maximal length between all names of this domain parameters.	

⌋(*RS_CRYPTO_02309*)

8.17.1.1.2.4 virtual ara::core::Result<std::size_t> ara::crypto::cryp::DomainParameters::GetParameterName (std::size_t *index*, ara::core::String ∗ *name* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20914]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::GetParameterName(std::size_t index, ara::core::String *name=nullptr)	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual ara::core::Result<std::size_t> GetParameterName (std::size_t index, ara::core::String *name=nullptr) const noexcept=0;`	
Parameters (in):	index	index of the parameter, which name is requested. It should be less than GetParametersCount()
Parameters (out):	name	an optional pointer to a string, where the parameter's name should be saved
Return value:	ara::core::Result< std::size_t >	actual length of the parameter name
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Get the parameter name by its index.	
Notes:	Capacity of the provided target string must be enough for the parameter's name saving. A list of supported parameters is specific for Crypto Primitive implementation and should be defined by a supplier. [Error]: SecurityErrorDomain::kUnknownIdentifier if the index argument is incorrect [Error]: SecurityErrorDomain::kInsufficientCapacity if (name != nullptr), but the name->capacity() is not enough to store the parameter name	

⌋(*RS_CRYPTO_02309*)

8.17.1.1.2.5 virtual ara::core::Result<std::size_t> ara::crypto::cryp::DomainParameters::ExpectedParameterSize (std::size_t *index*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20915]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::ExpectedParameterSize(std::size_t index)	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual ara::core::Result<std::size_t> ExpectedParameterSize (std::size_t index) const noexcept=0;`	
Parameters (in):	index	index of the parameter that should be set. It should be less than GetParametersCount()
Return value:	ara::core::Result< std::size_t >	expected (presize or maximal) size of the parameter defined by the index
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Get expected size of specific parameter of the Crypto Primitive.	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if the index argument is incorrect	

⌋*(RS_CRYPTO_02309)*

8.17.1.1.2.6 virtual ara::core::Result<void> ara::crypto::cryp::DomainParameters::SetParameter (std::size_t *index,* ReadOnlyMemRegion *value*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20916]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::SetParameter(std::size_t index, ReadOnlyMemRegion value)	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual ara::core::Result<void> SetParameter (std::size_t index, ReadOnlyMemRegion value) noexcept=0;`	
Parameters (in):	index	index of the parameter, which should be set (it should be less than GetParametersCount())
	value	a new value of the parameter that should be set
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Set a value to specific parameter of the Crypto Primitive.	

▽

△

Notes:	All Crypto Primitives that supports custom parameters also should have a correct default set of parameters. A list of supported parameters is specific for Crypto Primitive implementation and should be defined by a supplier. All named domain parameters (for which GetUniqueName() returns non-empty string) are already created in the completed state and don't need a call of this method! [Error]: SecurityErrorDomain::kUnknownIdentifier if the index argument is incorrect [Error]: SecurityErrorDomain::kUnexpectedValue if the value argument is incorrect [Error]: SecurityErrorDomain::kLogicFault if the domain parameters' set is already completed

⌋*(RS_CRYPTO_02309)*

8.17.1.1.2.7 virtual bool ara::crypto::cryp::DomainParameters::IsCompleted ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20917]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::IsCompleted()	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual bool IsCompleted () const noexcept=0;`	
Return value:	bool	true if this set of domain parameters is completed and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Check completeness and consistency of this parameters set.	
Notes:	Until the set of domain parameters is incomplete the object's COUID should not be set (i.e. it should be zero)!	

⌋*(RS_CRYPTO_02311)*

8.17.1.1.2.8 virtual bool ara::crypto::cryp::DomainParameters::Complete (Usage *allowedUsage*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20918]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::DomainParameters::Complete(Usage allowedUsage)
Scope:	class ara::crypto::cryp::DomainParameters
Syntax:	`virtual bool Complete (Usage allowedUsage) noexcept=0;`

▽

△

Parameters (in):	allowedUsage	a combination of bit-flags that specifies allowed usages of the domain parameters
Return value:	bool	true if this set of domain parameters is completed and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Check completeness and consistency of this parameters set and generates the COUID if the parameters are completed at first time.	
Notes:	Until the set of domain parameters is incomplete the object's COUID should not be set (i.e. it should be zero)! If the set of domain parameters is already completed then all following calls of the method Set Parameter() will fail! All named domain parameters (for which GetUniqueName() returns non-empty string) are already created in the completed state and have assigned COUID! Internal presence of a non-zero COUID is a marker of completed domain parameters, therefore the real check for completeness is required only if the COUID is zeros.	

⌋*(RS_CRYPTO_02311)*

8.17.1.1.2.9 virtual ara::core::StringView ara::crypto::cryp::DomainParameters::GetUniqueName ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_20919]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::DomainParameters::GetUniqueName()	
Scope:	class ara::crypto::cryp::DomainParameters	
Syntax:	`virtual ara::core::StringView GetUniqueName () const noexcept=0;`	
Return value:	ara::core::StringView	unique name of this domain parameters set or the empty string if a unique name is not assigned
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/domain_parameters.h"	
Description:	Return a unique "well-known" name of the parameters set, if it is assigned (i.e. OID/Name).	
Notes:	If this method returns a non-empty string (it is actual for all "Named" domain parameters) then it is already in the completed state and has an assigned COUID! The life-time of the returned StringView instance should not exceed the life-time of this Domain Parameters instance!	

⌋*(RS_CRYPTO_02309)*

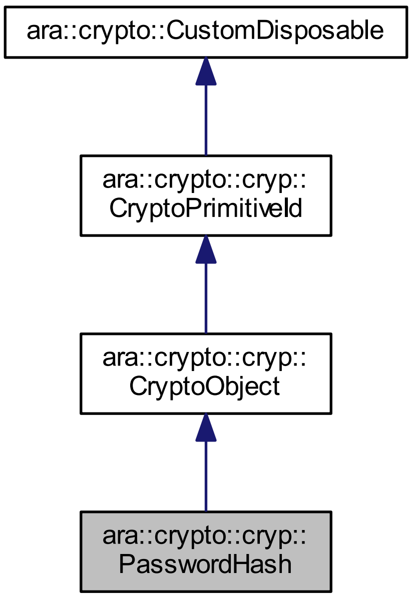
8.17.1.2 class ara::crypto::cryp::PasswordHash

[SWS_CRYPT_22400]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::PasswordHash
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::CryptoObject
Syntax:	`class PasswordHash : public CryptoObject {...};`
Header file:	#include "ara/crypto/cryp/password_hash.h"
Description:	Secure Password Hash object interface.
Notes:	The password hash object cannot be exportable! This object includes "seed" (randomization vector) and hash value of the password and seed.

⌋*(RS_CRYPTO_02106)*

Inheritance diagram for ara::crypto::cryp::PasswordHash:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoObject
ara::crypto::cryp:: PasswordHash

Public Types

- using Uptr = std::unique_ptr< PasswordHash, CustomDeleter >

Public Member Functions

- virtual std::size_t GetHashSize () const noexcept=0
- virtual std::size_t GetSeedSize () const noexcept=0

Additional Inherited Members

8.17.1.2.1 Member Typedef Documentation

8.17.1.2.1.1 using ara::crypto::cryp::PasswordHash::Uptr = std::unique_ptr<PasswordHash, CustomDeleter>

[SWS_CRYPT_22401]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::PasswordHash::Uptr
Scope:	class ara::crypto::cryp::PasswordHash
Derived from:	std::unique_ptr<PasswordHash, CustomDeleter>
Syntax:	`using ara::crypto::cryp::PasswordHash::Uptr = std::unique_ptr<PasswordHash, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/password_hash.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.17.1.2.2 Member Function Documentation

8.17.1.2.2.1 virtual std::size_t ara::crypto::cryp::PasswordHash::GetHashSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_22411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordHash::GetHashSize()	
Scope:	class ara::crypto::cryp::PasswordHash	
Syntax:	`virtual std::size_t GetHashSize () const noexcept=0;`	
Return value:	std::size_t	size of the hash value in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_hash.h"	
Description:	Get the hash size of this object.	

⌋*(RS_CRYPTO_02309)*

8.17.1.2.2.2 virtual std::size_t ara::crypto::cryp::PasswordHash::GetSeedSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_22412]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordHash::GetSeedSize()	
Scope:	class ara::crypto::cryp::PasswordHash	
Syntax:	`virtual std::size_t GetSeedSize () const noexcept=0;`	
Return value:	std::size_t	size of the seed value in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_hash.h"	
Description:	Get the seed size of this object.	

⌋*(RS_CRYPTO_02309)*

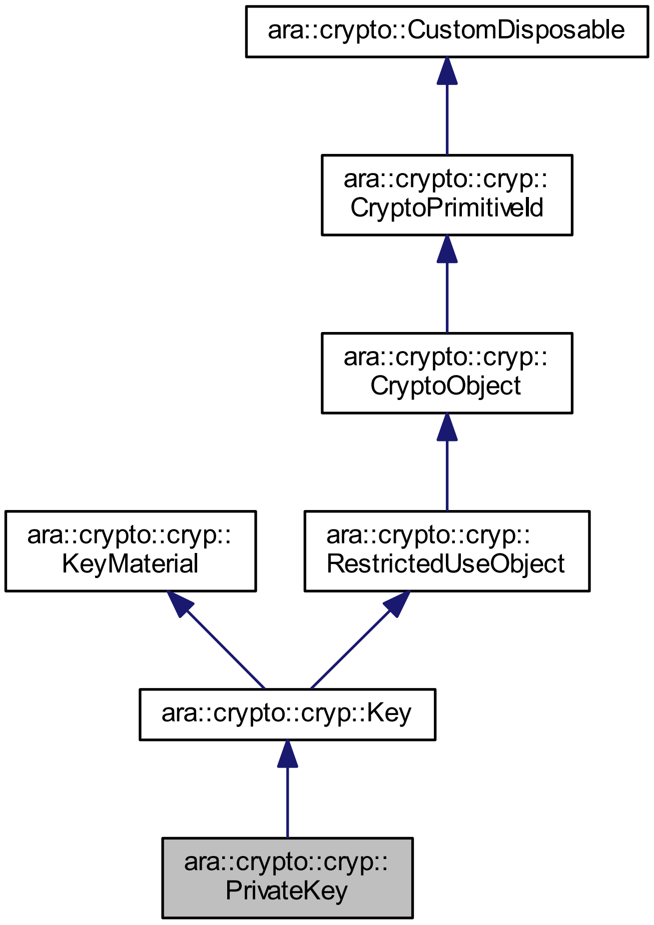
8.17.1.3 class ara::crypto::cryp::PrivateKey

[SWS_CRYPT_22500]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02002, RS_CRYPTO_02403)*

Inheritance diagram for ara::crypto::cryp::PrivateKey:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoObject
ara::crypto::cryp::KeyMaterial
ara::crypto::cryp::RestrictedUseObject
ara::crypto::cryp::Key
ara::crypto::cryp::PrivateKey

Public Types

- using Uptrc = std::unique_ptr< const PrivateKey, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< PublicKey::Uptrc > GetPublicKey (DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

Static Public Member Functions

- static ara::core::Result< PrivateKey::Uptrc > Cast (Key::Uptrc &&key) noexcept

Additional Inherited Members

8.17.1.3.1 Member Typedef Documentation

8.17.1.3.1.1 using ara::crypto::cryp::PrivateKey::Uptrc = std::unique_ptr<const PrivateKey, CustomDeleter>

[SWS_CRYPT_22501]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::PrivateKey::Uptrc
Scope:	class ara::crypto::cryp::PrivateKey
Derived from:	std::unique_ptr<const PrivateKey, CustomDeleter>
Syntax:	`using ara::crypto::cryp::PrivateKey::Uptrc = std::unique_ptr<const PrivateKey, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/private_key.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.17.1.3.2 Member Function Documentation

8.17.1.3.2.1 virtual ara::core::Result<PublicKey::Uptrc> ara::crypto::cryp::PrivateKey::GetPublicKey (DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22511]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PrivateKey::GetPublicKey(DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::PrivateKey	
Syntax:	`virtual ara::core::Result<PublicKey::Uptrc> GetPublicKey (DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	params	an optional pointer to domain parameters required for full specification of the transformation

▽

△

	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
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/private_key.h"	
Description:	Get the public key correspondent to this private key.	
Notes:	[Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type (incompatible with this algorithm) [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has incompleted state [Error]: SecurityErrorDomain::kUnexpectedValue if provided domain parameters object has COUID different from the COUID referenced in this private key (i.e. returned by this->Has Dependence()) [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02108, RS_CRYPTO_02115, RS_CRYPTO_02404)*

8.17.1.3.2.2 ara::core::Result< PrivateKey::Uptrc > ara::crypto::cryp::PrivateKey::Cast (Key::Uptrc && *key*) `[static], [noexcept]`

[SWS_CRYPT_22512]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PrivateKey::Cast(Key::Uptrc &&key)	
Scope:	class ara::crypto::cryp::PrivateKey	
Syntax:	`inline ara::core::Result< PrivateKey::Uptrc > Cast (Key::Uptrc &&key) noexcept;`	
Parameters (in):	key	unique smart pointer to the constant Key interface
Return value:	ara::core::Result< PrivateKey::Uptrc >	unique smart pointer to downcasted constant interface PrivateKey
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/private_key.h"	
Description:	Downcast and move unique smart pointer of the base Key interface to the PrivateKey one.	

▽

△

Notes:	[Error]: SecurityErrorDomain::kBadObjectType if an actual type of the key is not PrivateKey::Uptrc

⌋(*RS_CRYPTO_02404*, *RS_CRYPTO_02311*)

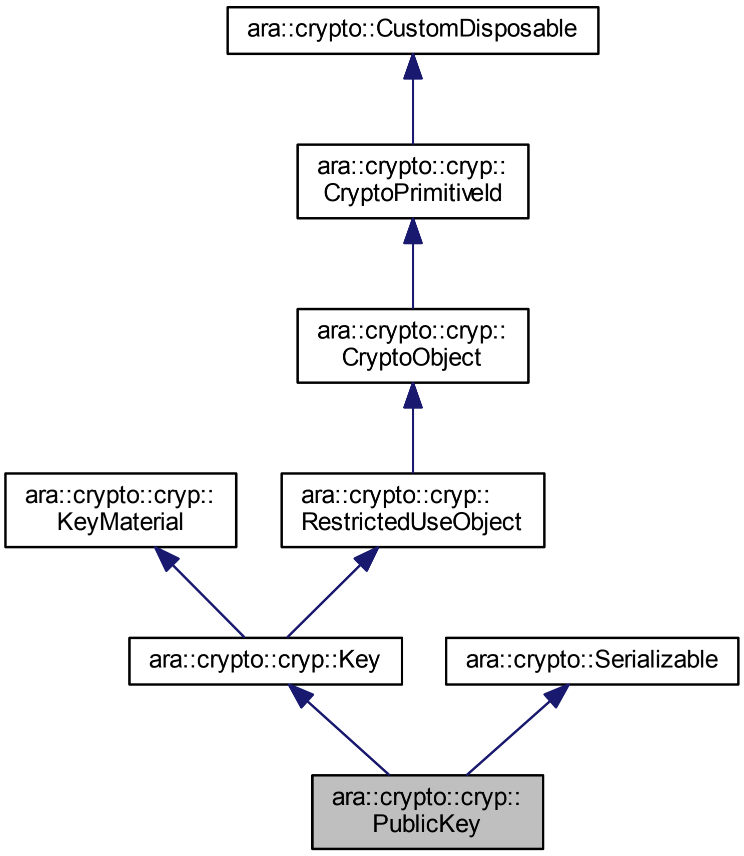
8.17.1.4 class ara::crypto::cryp::PublicKey

[SWS_CRYPT_22700]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02403*)

Inheritance diagram for ara::crypto::cryp::PublicKey:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoObject
ara::crypto::cryp::KeyMaterial
ara::crypto::cryp::RestrictedUseObject
ara::crypto::cryp::Key
ara::crypto::Serializable
ara::crypto::cryp::PublicKey

Public Types

- using Uptrc = std::unique_ptr< const PublicKey, CustomDeleter >

Public Member Functions

- virtual bool CheckKey (bool strongCheck=true) const noexcept=0
- virtual ara::core::Result< std::size_t > HashPublicKey (WritableMemRegion hash, HashFunctionCtx &hashFunc) const noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< void > HashPublicKey (ByteVectorT< Alloc > &hash, HashFunctionCtx &hashFunc) const noexcept

Static Public Member Functions

- static ara::core::Result< PublicKey::Uptrc > Cast (Key::Uptrc &&key) noexcept

Additional Inherited Members

8.17.1.4.1 Member Typedef Documentation

8.17.1.4.1.1 using ara::crypto::cryp::PublicKey::Uptrc = std::unique_ptr<const PublicKey, CustomDeleter>

[SWS_CRYPT_22701]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::PublicKey::Uptrc
Scope:	class ara::crypto::cryp::PublicKey
Derived from:	std::unique_ptr<const PublicKey, CustomDeleter>
Syntax:	`using ara::crypto::cryp::PublicKey::Uptrc = std::unique_ptr<const PublicKey, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/public_key.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.17.1.4.2 Member Function Documentation

8.17.1.4.2.1 virtual bool ara::crypto::cryp::PublicKey::CheckKey (bool *strongCheck* = `true`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22711]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PublicKey::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/public_key.h"	
Description:	Check the key for its correctness.	

⌋*(RS_CRYPTO_02311)*

8.17.1.4.2.2 virtual ara::core::Result<std::size_t> ara::crypto::cryp::PublicKey::HashPublicKey (WritableMemRegion *hash*, HashFunctionCtx & *hashFunc*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22712]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02311)*

8.17.1.4.2.3 template<typename Alloc = DefBytesAllocator> ara::core::Result<void> ara::crypto::cryp::PublicKey::HashPublicKey (ByteVectorT< Alloc > & *hash*, HashFunctionCtx & *hashFunc*) const `[noexcept]`

[SWS_CRYPT_22713]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PublicKey::HashPublicKey(ByteVectorT< Alloc > &hash, HashFunctionCtx &hashFunc)	
Scope:	class ara::crypto::cryp::PublicKey	

Syntax:	`template <typename Alloc>`
	`inline ara::core::Result<void> HashPublicKey (ByteVectorT< Alloc > &hash, 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

▽

△

Parameters (out):	hash	pre-reserved managed container for the resulting hash value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/public_key.h"	
Description:	Calculate hash of the Public Key value.	
Notes:	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. [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the hash buffer is not enough for storing of the result [Error]: SecurityErrorDomain::kIncompleteArgState if the hashFunc context is not initialized by required domain parameters	

⌋*(RS_CRYPTO_02311)*

8.17.1.4.2.4 ara::core::Result< PublicKey::Uptrc > ara::crypto::cryp::PublicKey::Cast (Key::Uptrc && *key*) `[static]`, `[noexcept]`

[SWS_CRYPT_22714]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PublicKey::Cast(Key::Uptrc &&key)	
Scope:	class ara::crypto::cryp::PublicKey	
Syntax:	`inline ara::core::Result< PublicKey::Uptrc > Cast (Key::Uptrc &&key) noexcept;`	
Parameters (in):	key	unique smart pointer to the constant Key interface
Return value:	ara::core::Result< PublicKey::Uptrc >	unique smart pointer to downcasted constant interface PublicKey
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/public_key.h"	
Description:	Downcast and move unique smart pointer of the base Key interface to the PublicKey one.	
Notes:	[Error]: SecurityErrorDomain::kBadObjectType if an actual type of the key is not PublicKey::Uptrc	

⌋*(RS_CRYPTO_02404, RS_CRYPTO_02311)*

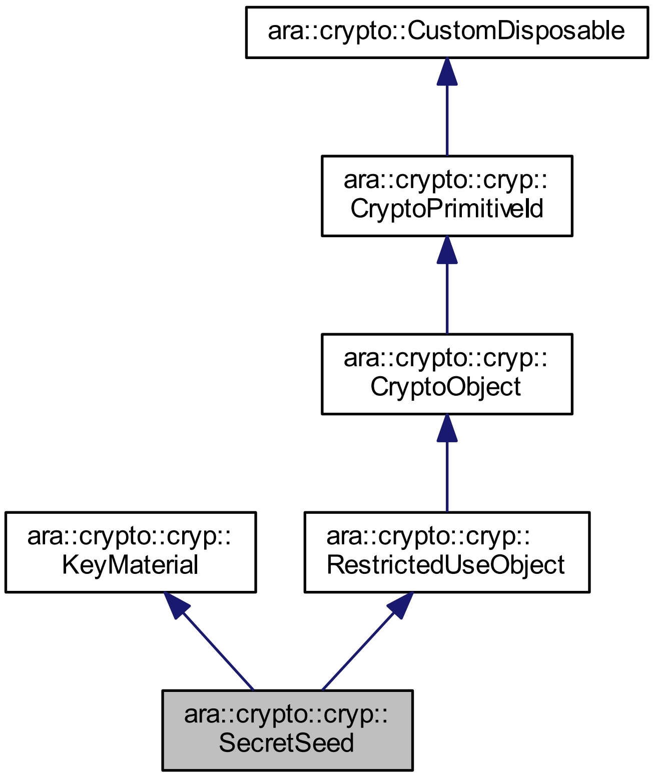
8.17.1.5 class ara::crypto::cryp::SecretSeed

[SWS_CRYPT_23000]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::SecretSeed
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::KeyMaterial
Syntax:	`class SecretSeed : public KeyMaterial {...};`
Header file:	#include "ara/crypto/cryp/secret_seed.h"
Description:	Secret Seed object interface.
Notes:	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)*

Inheritance diagram for ara::crypto::cryp::SecretSeed:



Public Types

- using Uptrc = std::unique_ptr< const SecretSeed, CustomDeleter >
- using Uptr = std::unique_ptr< SecretSeed, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< SecretSeed::Uptr > Clone (ReadOnlyMemRegion xorDelta=ReadOnlyMemRegion(), ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0
- virtual ara::core::Result< SecretSeed & > JumpFrom (const SecretSeed &from, std::int64_t steps) noexcept=0
- virtual SecretSeed & Next () noexcept=0
- virtual SecretSeed & Jump (std::int64_t steps) noexcept=0
- virtual SecretSeed & operator^= (const SecretSeed &source) noexcept=0
- virtual SecretSeed & operator^= (ReadOnlyMemRegion source) noexcept=0

Additional Inherited Members

8.17.1.5.1 Member Typedef Documentation

8.17.1.5.1.1 using ara::crypto::cryp::SecretSeed::Uptrc = std::unique_ptr<const SecretSeed, CustomDeleter>

[SWS_CRYPT_23001]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::SecretSeed::Uptrc
Scope:	class ara::crypto::cryp::SecretSeed
Derived from:	std::unique_ptr<const SecretSeed, CustomDeleter>
Syntax:	`using ara::crypto::cryp::SecretSeed::Uptrc = std::unique_ptr<const SecretSeed, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/secret_seed.h"
Description:	Unique smart pointer of a constant interface instance.

⌋*(RS_CRYPTO_02404)*

8.17.1.5.1.2 using ara::crypto::cryp::SecretSeed::Uptr = std::unique_ptr<SecretSeed, CustomDeleter>

[SWS_CRYPT_23002]{DRAFT} ⌈

Kind:	type alias	
Symbol:	ara::crypto::cryp::SecretSeed::Uptr	
Scope:	class ara::crypto::cryp::SecretSeed	
Derived from:	std::unique_ptr<SecretSeed, CustomDeleter>	
Syntax:	`using ara::crypto::cryp::SecretSeed::Uptr = std::unique_ptr<SecretSeed, CustomDeleter>;`	
Header file:	#include "ara/crypto/cryp/secret_seed.h"	
Description:	Unique smart pointer of a volatile interface instance.	

⌋(*RS_CRYPTO_02404*)

8.17.1.5.2 Member Function Documentation

8.17.1.5.2.1 virtual ara::core::Result<SecretSeed::Uptr> ara::crypto::cryp::SecretSeed::Clone (ReadOnlyMemRegion *xorDelta* = ReadOnlyMemRegion `()`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23011]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SecretSeed::Clone(ReadOnlyMemRegion xorDelta=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::SecretSeed	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptr> Clone (ReadOnlyMemRegion xorDelta=ReadOnlyMemRegion(), ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	xorDelta	optional "delta" value that must be XOR-ed with the "cloned" copy of the original seed
	reservedIndex	the optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
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/secret_seed.h"	
Description:	Clone this Secret Seed object to new session object.	
Notes:	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. [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02311)*

8.17.1.5.2.2 virtual ara::core::Result<SecretSeed&> ara::crypto::cryp::SecretSeed::JumpFrom (const SecretSeed & *from*, std::int64_t *steps*) `[pure virtual], [noexcept]`

[SWS_CRYPT_23012]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SecretSeed::JumpFrom(const SecretSeed &from, std::int64_t steps)	
Scope:	class ara::crypto::cryp::SecretSeed	
Syntax:	`virtual ara::core::Result<SecretSeed&> 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< SecretSeed & >	reference to this updated object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/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.	
Notes:	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! [Error]: SecurityErrorDomain::kIncompatibleObject if this object and the from argument are associated with incompatible cryptographic algorithms [Error]: SecurityErrorDomain::kInvalidInputSize if value size of the from seed is less then value size of this one	

⌋*(RS_CRYPTO_02311)*

8.17.1.5.2.3 virtual SecretSeed& ara::crypto::cryp::SecretSeed::Next ( ) `[pure virtual], [noexcept]`

[SWS_CRYPT_23013]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SecretSeed::Next()	
Scope:	class ara::crypto::cryp::SecretSeed	
Syntax:	`virtual SecretSeed& Next () noexcept=0;`	
Return value:	SecretSeed &	reference to this updated object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/secret_seed.h"	
Description:	Set next value of the secret seed according to "counting" expression defined by a cryptographic algorithm associated with this object.	
Notes:	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_02311*)

8.17.1.5.2.4 virtual SecretSeed& ara::crypto::cryp::SecretSeed::Jump (std::int64_t *steps*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23014]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SecretSeed::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/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.	
Notes:	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_02311*)

8.17.1.5.2.5 virtual SecretSeed& ara::crypto::cryp::SecretSeed::operator^= (const SecretSeed & *source*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23015]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SecretSeed::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/secret_seed.h"	
Description:	XOR value of this seed object with another one and save result to this object.	
Notes:	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_02311)*

8.17.1.5.2.6 virtual SecretSeed& ara::crypto::cryp::SecretSeed::operator^= (ReadOnlyMemRegion *source*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23016]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SecretSeed::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/secret_seed.h"	
Description:	XOR value of this seed object with provided memory region and save result to this object.	
Notes:	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_02311)*

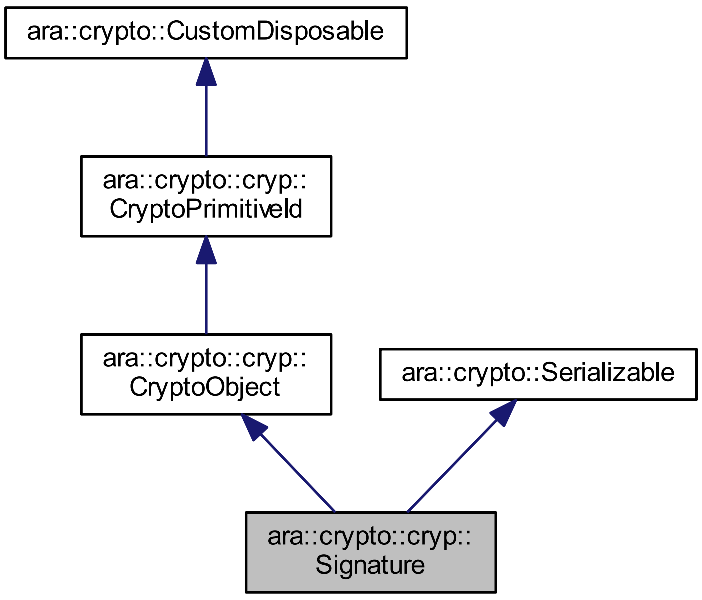
8.17.1.6 class ara::crypto::cryp::Signature

[SWS_CRYPT_23300]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::Signature
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::CryptoObject
Syntax:	`class Signature : public CryptoObject {...};`
Header file:	#include "ara/crypto/cryp/signature.h"
Description:	Signature container interface
Notes:	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)*

Inheritance diagram for ara::crypto::cryp::Signature:



Public Types

- using Uptrc = std::unique_ptr< const Signature, CustomDeleter >

Public Member Functions

- virtual CryptoPrimitiveId::AlgId GetHashAlgId () const noexcept=0

Protected Member Functions

- virtual ∼Signature () noexcept=default

Additional Inherited Members

8.17.1.6.1 Member Typedef Documentation

8.17.1.6.1.1 using ara::crypto::cryp::Signature::Uptrc = std::unique_ptr<const Signature, CustomDeleter>

[SWS_CRYPT_23301]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::Signature::Uptrc
Scope:	class ara::crypto::cryp::Signature
Derived from:	std::unique_ptr<const Signature, CustomDeleter>
Syntax:	`using ara::crypto::cryp::Signature::Uptrc = std::unique_ptr<const Signature, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/signature.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.17.1.6.2 Constructor & Destructor Documentation

8.17.1.6.2.1 virtual ara::crypto::cryp::Signature::∼Signature ( ) [protected], [virtual], [default], [noexcept]

[SWS_CRYPT_23310]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02311)*

8.17.1.6.3 Member Function Documentation

8.17.1.6.3.1 virtual CryptoPrimitiveId::AlgId ara::crypto::cryp::Signature::GetHashAlgId ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23311]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::Signature::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/signature.h"	
Description:	Get an ID of hash algorithm used for this signature object production.	

⌋*(RS_CRYPTO_02311)*

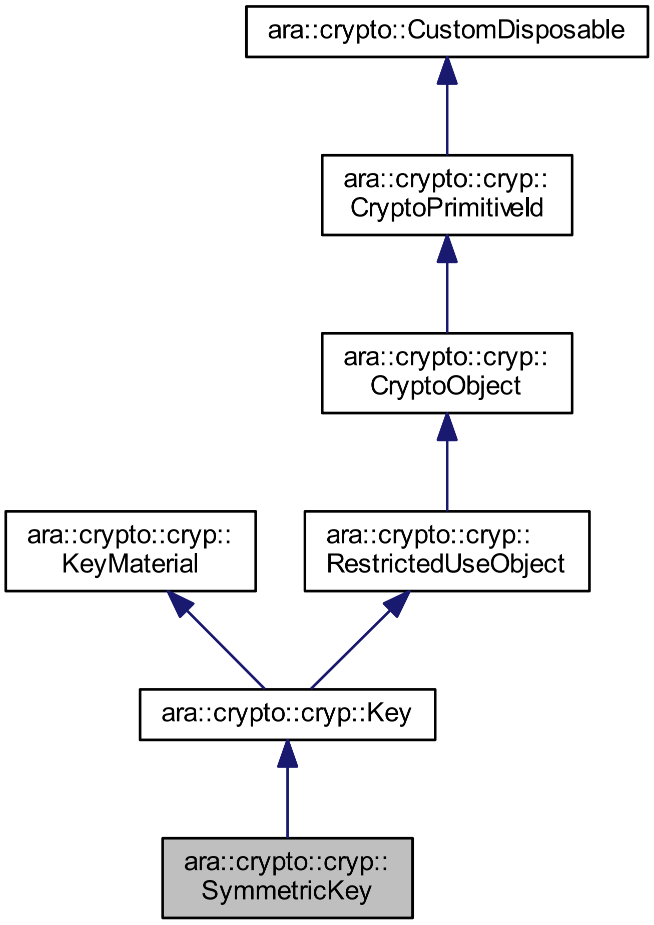
8.17.1.7 class ara::crypto::cryp::SymmetricKey

[SWS_CRYPT_23800]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02001, RS_CRYPTO_02403)*

Inheritance diagram for ara::crypto::cryp::SymmetricKey:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoObject
ara::crypto::cryp:: KeyMaterial
ara::crypto::cryp:: RestrictedUseObject
ara::crypto::cryp::Key
ara::crypto::cryp:: SymmetricKey

Public Types

- using Uptrc = std::unique_ptr< const SymmetricKey, CustomDeleter >

Static Public Member Functions

- static ara::core::Result< SymmetricKey::Uptrc > Cast (Key::Uptrc &&key) noexcept

Additional Inherited Members

8.17.1.7.1 Member Typedef Documentation

8.17.1.7.1.1 using ara::crypto::cryp::SymmetricKey::Uptrc = std::unique_ptr<const SymmetricKey, CustomDeleter>

[SWS_CRYPT_23801]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::SymmetricKey::Uptrc
Scope:	class ara::crypto::cryp::SymmetricKey
Derived from:	std::unique_ptr<const SymmetricKey, CustomDeleter>
Syntax:	`using ara::crypto::cryp::SymmetricKey::Uptrc = std::unique_ptr<const SymmetricKey, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/symmetric_key.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.17.1.7.2 Member Function Documentation

8.17.1.7.2.1 ara::core::Result< SymmetricKey::Uptrc > ara::crypto::cryp::SymmetricKey::Cast (Key::Uptrc && *key*) `[static]`, `[noexcept]`

[SWS_CRYPT_23811]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKey::Cast(Key::Uptrc &&key)	
Scope:	class ara::crypto::cryp::SymmetricKey	
Syntax:	`inline ara::core::Result< SymmetricKey::Uptrc > Cast (Key::Uptrc &&key) noexcept;`	
Parameters (in):	key	unique smart pointer to the constant Key interface
Return value:	ara::core::Result< SymmetricKey::Uptrc >	unique smart pointer to downcasted constant interface SymmetricKey
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key.h"	
Description:	Downcast and move unique smart pointer of the base Key interface to the SymmetricKey one.	
Notes:	[Error]: SecurityErrorDomain::kBadObjectType if an actual type of the key is not SymmetricKey::Uptrc	

⌋*(RS_CRYPTO_02404, RS_CRYPTO_02311)*

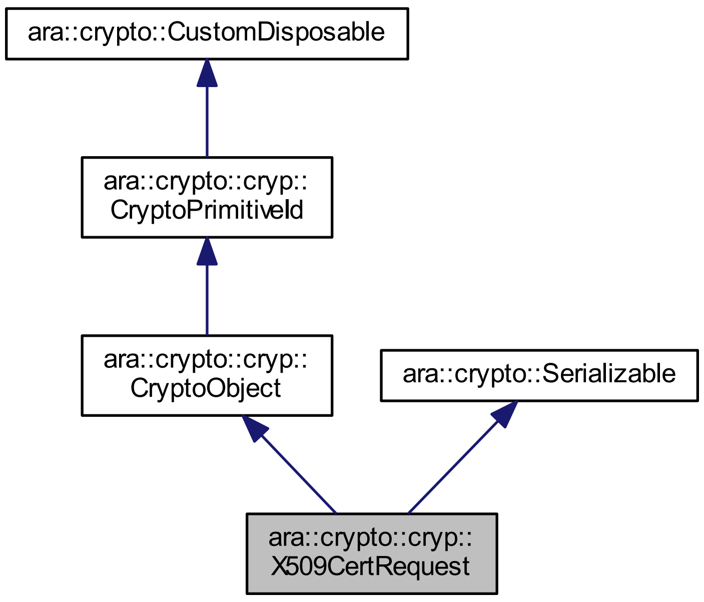
8.17.1.8 class ara::crypto::cryp::X509CertRequest

[SWS_CRYPT_24300]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::X509CertRequest
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::CryptoObject
Syntax:	`class X509CertRequest : public CryptoObject {...};`
Header file:	#include "ara/crypto/cryp/x509_cert_request.h"
Description:	Simple interface of the Certification Request object.
Notes:	This interface is not dedicated for complete parsing of Certification Requests. It is dedicated only for decoding of the cryptographically essential part only (subject public key and signature), reqired for creation and cryptographic validation of the request. Other interface x509::CertSign Request is dedicated for complete parsing of the request content.

⌋*(RS_CRYPTO_02307)*

Inheritance diagram for ara::crypto::cryp::X509CertRequest:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoObject
ara::crypto::Serializable
ara::crypto::cryp:: X509CertRequest

Public Types

- using Uptrc = std::unique_ptr< const X509CertRequest >

Public Member Functions

- virtual ara::core::Result< bool > Verify (HashFunctionCtx &hash, VerifierPublicCtx &verifier) const noexcept=0
- virtual unsigned Version () const noexcept=0
- virtual const X509Signature & Signature () const noexcept=0
- virtual const X509PublicKeyInfo & SubjectPublicKeyInfo () const noexcept=0

Additional Inherited Members

8.17.1.8.1 Member Typedef Documentation

8.17.1.8.1.1 using ara::crypto::cryp::X509CertRequest::Uptrc = std::unique_ptr<const X509CertRequest>

[SWS_CRYPT_24301]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::X509CertRequest::Uptrc
Scope:	class ara::crypto::cryp::X509CertRequest
Derived from:	std::unique_ptr<const X509CertRequest>
Syntax:	`using ara::crypto::cryp::X509CertRequest::Uptrc = std::unique_ptr<const X509CertRequest>;`
Header file:	#include "ara/crypto/cryp/x509_cert_request.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.17.1.8.2 Member Function Documentation

8.17.1.8.2.1 virtual ara::core::Result<bool> ara::crypto::cryp::X509CertRequest::Verify (HashFunctionCtx & *hash*, VerifierPublicCtx & *verifier*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24311]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::X509CertRequest::Verify(HashFunctionCtx &hash, VerifierPublicCtx &verifier)
Scope:	class ara::crypto::cryp::X509CertRequest
Syntax:	`virtual ara::core::Result<bool> Verify (HashFunctionCtx &hash, VerifierPublicCtx &verifier) const noexcept=0;`

▽

△

Parameters (in):	hash	a temporary hash-function context that should be used in the call (the Alg ID can be got as X509Signature::GetPrimitiveId())
	verifier	a temporary signature verification context that should be used in the call (the Alg ID can be got as X509Signature::GetRequiredHashAlgId())
Return value:	ara::core::Result< bool >	true if the signature is correct and @ false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_cert_request.h"	
Description:	Verify self-signed signature of the certificate request.	
Notes:	This method uses key values and domain parameters stored inside the object! [Error]: SecurityErrorDomain::kIncompatibleObject if the hash or verifier arguments are configured for algorithms different from the one applied for signature of this certification request	

⌋*(RS_CRYPTO_02306)*

8.17.1.8.2.2 virtual unsigned ara::crypto::cryp::X509CertRequest::Version ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_24312]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509CertRequest::Version()	
Scope:	class ara::crypto::cryp::X509CertRequest	
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/cryp/x509_cert_request.h"	
Description:	Return format version of the certificate request.	

⌋*(RS_CRYPTO_02311)*

8.17.1.8.2.3 virtual const X509Signature& ara::crypto::cryp::X509CertRequest::Signature ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_24313]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509CertRequest::Signature()	
Scope:	class ara::crypto::cryp::X509CertRequest	
Syntax:	`virtual const X509Signature& Signature () const noexcept=0;`	
Return value:	const X509Signature &	signature object of the request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_cert_request.h"	
Description:	Return signature object of the request.	

⌋*(RS_CRYPTO_02306)*

8.17.1.8.2.4 virtual const X509PublicKeyInfo& ara::crypto::cryp::X509CertRequest::SubjectPublicKeyInfo ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24314]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509CertRequest::SubjectPublicKeyInfo()	
Scope:	class ara::crypto::cryp::X509CertRequest	
Syntax:	`virtual const X509PublicKeyInfo& SubjectPublicKeyInfo () const noexcept=0;`	
Return value:	const X509PublicKeyInfo &	subject public key included to the request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_cert_request.h"	
Description:	Return subject public key included to the request.	

⌋*(RS_CRYPTO_02306)*

8.17.1.9 class ara::crypto::cryp::X509PublicKeyInfo

[SWS_CRYPT_24400]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::X509PublicKeyInfo
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::X509AlgorithmId
Syntax:	`class X509PublicKeyInfo : public X509AlgorithmId {...};`
Header file:	#include "ara/crypto/cryp/x509_public_key_info.h"

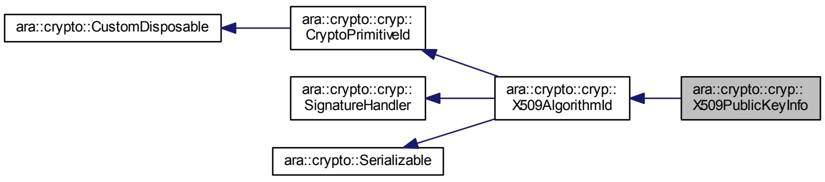
▽

△

Description:	X.509 Public Key Information interface.

⌋*(RS_CRYPTO_02307)*

Inheritance diagram for ara::crypto::cryp::X509PublicKeyInfo:



Public Types

- using Uptrc = std::unique_ptr< const X509PublicKeyInfo, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< PublicKey::Uptrc > GetPublicKey (DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0
- virtual bool IsSameKey (const PublicKey &publicKey) const noexcept=0

Additional Inherited Members

8.17.1.9.1 Member Typedef Documentation

8.17.1.9.1.1 using ara::crypto::cryp::X509PublicKeyInfo::Uptrc = std::unique_ptr<const X509PublicKeyInfo, CustomDeleter>

[SWS_CRYPT_24401]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::X509PublicKeyInfo::Uptrc
Scope:	class ara::crypto::cryp::X509PublicKeyInfo

▽

△

Derived from:	std::unique_ptr<const X509PublicKeyInfo, CustomDeleter>
Syntax:	`using ara::crypto::cryp::X509PublicKeyInfo::Uptrc = std::unique_ptr<const X509PublicKeyInfo, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/x509_public_key_info.h"
Description:	Unique smart pointer of the interface.

⌋(*RS_CRYPTO_02404*)

8.17.1.9.2 Member Function Documentation

8.17.1.9.2.1 virtual ara::core::Result<PublicKey::Uptrc> ara::crypto::cryp::X509PublicKeyInfo::GetPublicKey (DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509PublicKeyInfo::GetPublicKey(DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::X509PublicKeyInfo	
Syntax:	`virtual ara::core::Result<PublicKey::Uptrc> GetPublicKey (DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	params	an optional pointer to domain parameters required for full specification of the transformation
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< 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/cryp/x509_public_key_info.h"	
Description:	Get public key object of the subject.	
Notes:	[Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02108, RS_CRYPTO_02306, RS_CRYPTO_02404)*

8.17.1.9.2.2 virtual bool ara::crypto::cryp::X509PublicKeyInfo::IsSameKey (const PublicKey & *publicKey*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24412]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509PublicKeyInfo::IsSameKey(const PublicKey &publicKey)	
Scope:	class ara::crypto::cryp::X509PublicKeyInfo	
Syntax:	`virtual bool IsSameKey (const 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/cryp/x509_public_key_info.h"	
Description:	Verify the sameness of the provided and kept public keys.	
Notes:	This method compare the public key values only.	

⌋*(RS_CRYPTO_02311)*

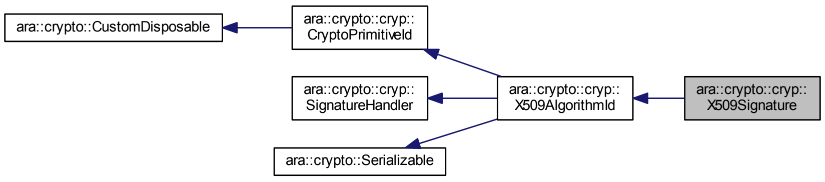
8.17.1.10 class ara::crypto::cryp::X509Signature

[SWS_CRYPT_24600]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::X509Signature
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::X509AlgorithmId
Syntax:	`class X509Signature : public X509AlgorithmId {...};`
Header file:	#include "ara/crypto/cryp/x509_signature.h"
Description:	X.509 Signature interface (X.509 Algorithm Identifier & Signature Value).

⌋*(RS_CRYPTO_02307)*

Inheritance diagram for ara::crypto::cryp::X509Signature:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::SignatureHandler
ara::crypto::cryp::X509AlgorithmId
ara::crypto::cryp::X509Signature
ara::crypto::Serializable

Public Types

- using Uptrc = std::unique_ptr< const X509Signature, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< Signature::Uptrc > GetSignature (ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

Additional Inherited Members

8.17.1.10.1 Member Typedef Documentation

8.17.1.10.1.1 using ara::crypto::cryp::X509Signature::Uptrc = std::unique_ptr<const X509Signature, CustomDeleter>

[SWS_CRYPT_24601]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::X509Signature::Uptrc
Scope:	class ara::crypto::cryp::X509Signature
Derived from:	std::unique_ptr<const X509Signature, CustomDeleter>
Syntax:	`using ara::crypto::cryp::X509Signature::Uptrc = std::unique_ptr<const X509Signature, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/x509_signature.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.17.1.10.2 Member Function Documentation

8.17.1.10.2.1 virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::X509Signature::GetSignature (ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24611]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509Signature::GetSignature(ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::X509Signature	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> GetSignature (ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Signature::Uptrc >	unique smart pointer to the created signature object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_signature.h"	
Description:	Get signature object of the hosted DER-structure.	
Notes:	[Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02306, RS_CRYPTO_02404)*

8.18 Top-level transformation interfaces

Modules

- Symmetric transformation interfaces
- Asymmetric transformation interfaces

Classes

- class ara::crypto::cryp::HashFunctionCtx
- class ara::crypto::cryp::KeyDerivationFunctionCtx
- class ara::crypto::cryp::PasswordCache
- class ara::crypto::cryp::RandomGeneratorCtx

Detailed Description

This group consists of top-level interfaces of cryptographic contexts available for consumer applications.

8.18.1 Class Documentation

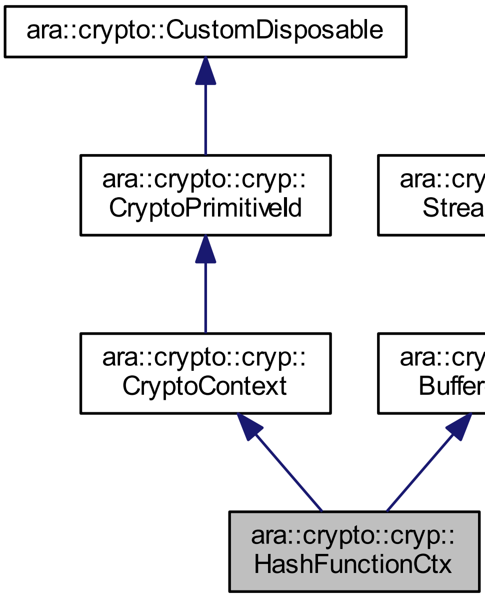
8.18.1.1 class ara::crypto::cryp::HashFunctionCtx

[SWS_CRYPT_21100]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02205)*

Inheritance diagram for ara::crypto::cryp::HashFunctionCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::StreamStarter
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::BufferedDigest
ara::crypto::cryp::HashFunctionCtx

Public Types

- using Uptr = std::unique_ptr< HashFunctionCtx, CustomDeleter >

Additional Inherited Members

8.18.1.1.1 Member Typedef Documentation

8.18.1.1.1.1 using ara::crypto::cryp::HashFunctionCtx::Uptr = std::unique_ptr<HashFunctionCtx, CustomDeleter>

[SWS_CRYPT_21101]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::HashFunctionCtx::Uptr
Scope:	class ara::crypto::cryp::HashFunctionCtx

▽

△

Derived from:	std::unique_ptr<HashFunctionCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::HashFunctionCtx::Uptr = std::unique_ptr<HashFunctionCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/hash_function_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

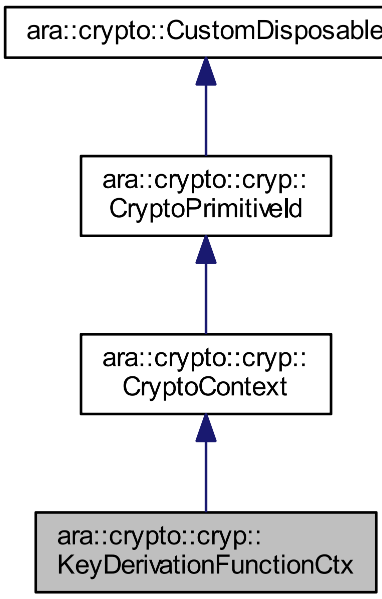
8.18.1.2 class ara::crypto::cryp::KeyDerivationFunctionCtx

[SWS_CRYPT_21500]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyDerivationFunctionCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::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)*

Inheritance diagram for ara::crypto::cryp::KeyDerivationFunctionCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyDerivationFunctionCtx

Public Types

- using Uptr = std::unique_ptr< KeyDerivationFunctionCtx, CustomDeleter >

Public Member Functions

- virtual std::uint32_t ConfigIterations (std::uint32_t iterations=0) noexcept=0
- virtual ara::core::Result< SymmetricKey::Uptrc > DeriveKey (AlgId targetAlgId, Key::Usage allowedUsage, const KeyMaterial &sourceKm, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), bool isSession=true, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, std::uint32_t iterations=0) const noexcept=0
- virtual ara::core::Result< SymmetricKey::Uptrc > DeriveKey (AlgId targetAlgId, Key::Usage allowedUsage, const KeyMaterial &sourceKm, const SecretSeed &salt, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), bool isSession=true, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, std::uint32_t iterations=0) const noexcept=0
- virtual ara::core::Result< SecretSeed::Uptrc > DeriveSeed (AlgId targetAlgId, SecretSeed::Usage allowedUsage, const KeyMaterial &sourceKm, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, std::uint32_t iterations=0) const noexcept=0
- virtual ara::core::Result< SecretSeed::Uptrc > DeriveSeed (AlgId targetAlgId, SecretSeed::Usage allowedUsage, const KeyMaterial &sourceKm, const SecretSeed &salt, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, std::uint32_t iterations=0) const noexcept=0

Additional Inherited Members

8.18.1.2.1 Member Typedef Documentation

8.18.1.2.1.1 using ara::crypto::cryp::KeyDerivationFunctionCtx::Uptr = std::unique_ptr<KeyDerivationFunctionCtx, CustomDeleter>

[SWS_CRYPT_21501]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::KeyDerivationFunctionCtx::Uptr
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx
Derived from:	std::unique_ptr<KeyDerivationFunctionCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::KeyDerivationFunctionCtx::Uptr = std::unique_ptr<KeyDerivationFunctionCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.18.1.2.2 Member Function Documentation

8.18.1.2.2.1 virtual std::uint32_t ara::crypto::cryp::KeyDerivationFunctionCtx::ConfigIterations (std::uint32_t *iterations* = *0*) [pure virtual], [noexcept]

[SWS_CRYPT_21511]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDerivationFunctionCtx::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 requred 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.	
Notes:	Implementation can restrict minimal and/or maximal value of the iterations number.	

⌋*(RS_CRYPTO_02309)*

8.18.1.2.2.2 virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveKey (AlgId *targetAlgId*, Key::Usage *allowedUsage*, const KeyMaterial & *sourceKm*, ReadOnlyMemRegion *salt* = ReadOnlyMemRegion `()`, ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion `()`, bool *isSession* = `true`, bool *isExportable* = `false`, DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap, std::uint32_t *iterations* = `0`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21512]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveKey(AlgId targetAlgId, Key::Usage allowed Usage, const KeyMaterial &sourceKm, ReadOnlyMemRegion salt=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> DeriveKey (AlgId target AlgId, Key::Usage allowedUsage, const KeyMaterial &sourceKm, ReadOnly MemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=Read OnlyMemRegion(), bool isSession=true, bool isExportable=false, Domain Parameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=k AllocObjectOnHeap, std::uint32_t iterations=0) const noexcept=0;`	
Parameters (in):	targetAlgId	the target symmetric algorithm identifier (also defines a target key-length)
	allowedUsage	the allowed usage scope of the target key
	sourceKm	the source key material for the key derivation execution
	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)
	isSession	the "session" (or "temporary") attribute for the target key (if true)
	isExportable	the exportability attribute for the target key (if true)
	params	an optional pointer to Domain Parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
	iterations	a requred number of base function iterations (0 means the current default number)
Return value:	ara::core::Result< Symmetric Key::Uptrc >	unique smart pointer to created instance of derived symetric key
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Derive a symmetric key from provided key material (with optional public salt).	

▽

△

Notes:	If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the produced key object. The byte sequence provided via argument ctxLabel can include a few fields with different meaning separated by single 0x00 byte. [Error]: SecurityErrorDomain::kBruteForceRisk if key length of the sourceKm is below of an internally defined limitation [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kIncompatibleObject if content (at least algorithm) of provided domain parameters is incompatible with targetAlgId [Error]: SecurityErrorDomain::kInvalidArgument if any of arguments is incorrect [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kUsageViolation if this transformation type is prohibited by the "allowed usage" restrictions of the provided sourceKm object

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02107, RS_CRYPTO_02108, RS_CRYPTO_02111, RS_CRYPTO_02113, RS_CRYPTO_02115, RS_CRYPTO_02404)*

8.18.1.2.2.3 virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveKey (AlgId *targetAlgId,* Key::Usage *allowedUsage,* const KeyMaterial & *sourceKm,* const SecretSeed & *salt,* ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion `()`, bool *isSession* = `true`, bool *isExportable* = `false`, DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap, std::uint32_t *iterations* = *0*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21513]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveKey(AlgId targetAlgId, Key::Usage allowed Usage, const KeyMaterial &sourceKm, const SecretSeed &salt, ReadOnlyMemRegion ctx Label=ReadOnlyMemRegion()

▽

△

Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> DeriveKey (AlgId targetAlgId, Key::Usage allowedUsage, const KeyMaterial &sourceKm, const SecretSeed &salt, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), bool isSession=true, bool isExportable=false, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, std::uint32_t iterations=0) const noexcept=0;`	
Parameters (in):	targetAlgId	the target symmetric algorithm identifier (also defines a target key-length)
	allowedUsage	the allowed usage scope of the target key
	sourceKm	the source key material for the key derivation execution
	salt	the secret salt value (an additional secret value independent from the sourceKm)
	ctxLabel	an optional application specific "context label" (it can identify purpose of the target key and/or communication parties)
	isSession	the "session" (or "temporary") attribute for the target key (if true)
	isExportable	the exportability attribute for the target key (if true)
	params	an optional pointer to Domain Parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
	iterations	a requred number of base function iterations (0 means the current default number)
Return value:	ara::core::Result< Symmetric Key::Uptrc >	unique smart pointer to created instance of derived symetric key
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Derive a symmetric key from provided key material (with secret salt).	
Notes:	If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the produced key object. The byte sequence provided via argument ctxLabel can include a few fields with different meaning separated by single 0x00 byte. [Error]: SecurityErrorDomain::kBruteForceRisk if key length of the sourceKm is below of an internally defined limitation [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kIncompatibleObject if content (at least algorithm) of provided domain parameters is incompatible with targetAlgId [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kUsageViolation if this transformation type is prohibited by the "allowed usage" restrictions of the provided sourceKm or salt objects [Error]: SecurityErrorDomain::kInvalidArgument if any of arguments is incorrect, and the error condition is not covered by the cases described above

⌋(*RS_CRYPTO_02102*, *RS_CRYPTO_02107*, *RS_CRYPTO_02108*, *RS_CRYPTO_02111*, *RS_CRYPTO_02113*, *RS_CRYPTO_02115*, *RS_CRYPTO_02404*)

8.18.1.2.2.4 virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveSeed (AlgId *targetAlgId*, SecretSeed::Usage *allowedUsage*, const KeyMaterial & *sourceKm*, ReadOnlyMemRegion *salt* = ReadOnlyMemRegion`()`, ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion`()`, bool *isSession* `= true`, bool *isExportable* `= false`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap, std::uint32_t *iterations* = `0`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21514]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveSeed(AlgId targetAlgId, SecretSeed::Usage allowedUsage, const KeyMaterial &sourceKm, ReadOnlyMemRegion salt=ReadOnlyMem Region()	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> DeriveSeed (AlgId target AlgId, SecretSeed::Usage allowedUsage, const KeyMaterial &sourceKm, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctx Label=ReadOnlyMemRegion(), bool isSession=true, bool is Exportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOn Heap, std::uint32_t iterations=0) const noexcept=0;`	
Parameters (in):	targetAlgId	the target symmetric algorithm identifier (also defines a target seed-length)
	allowedUsage	the allowed usage scope of the target seed
	sourceKm	the source key material for the key derivation execution
	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)
	isSession	the "session" (or "temporary") attribute for the target key material (if true)
	isExportable	the exportability attribute for the target key material (if true)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
	iterations	a requred number of base function iterations (0 means the current default number)
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to created SecretSeed object, which keeps the derived key material
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Derive a "slave" key material (secret seed) from provided "master" key material (with optional public salt).	
Notes:	The byte sequence provided via argument ctxLabel can include a few fields with different meaning separated by single 0x00 byte. [Error]: SecurityErrorDomain::kBruteForceRisk if key length of the sourceKm is below of an internally defined limitation [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kUsageViolation if this transformation type is prohibited by the "allowed usage" restrictions of the provided sourceKm object [Error]: SecurityErrorDomain::kInvalidArgument if any of arguments is incorrect, and the error condition is not covered by the cases described above	

⌋*(RS_CRYPTO_02007, RS_CRYPTO_02113, RS_CRYPTO_02404)*

8.18.1.2.2.5 virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveSeed (AlgId *targetAlgId*, SecretSeed::Usage *allowedUsage*, const KeyMaterial & *sourceKm*, const SecretSeed & *salt*, ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion*()*, bool *isSession* = *true*, bool *isExportable* = *false*, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap, std::uint32_t *iterations* = *0*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21515]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDerivationFunctionCtx::DeriveSeed(AlgId targetAlgId, SecretSeed::Usage allowedUsage, const KeyMaterial &sourceKm, const SecretSeed &salt, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::KeyDerivationFunctionCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> DeriveSeed (AlgId targetAlgId, SecretSeed::Usage allowedUsage, const KeyMaterial &sourceKm, const SecretSeed &salt, ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, std::uint32_t iterations=0) const noexcept=0;`	
Parameters (in):	targetAlgId	the target symmetric algorithm identifier (also defines a target seed-length)
	allowedUsage	the allowed usage scope of the target seed
	sourceKm	the source key material for the key derivation execution
	salt	the secret salt value (an additional secret value independent from the sourceKm)
	ctxLabel	an optional application specific "context label" (it can identify purpose of the target key and/or communication parties)
	isSession	the "session" (or "temporary") attribute for the target key material (if true)
	isExportable	the exportability attribute for the target key material (if true)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
	iterations	a requred number of base function iterations (0 means the current default number)
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to created SecretSeed object, which keeps the derived key material
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_derivation_function_ctx.h"	
Description:	Derive a "slave" key material (secret seed) from provided "master" key material (with secret salt).	
Notes:	The byte sequence provided via argument ctxLabel can include a few fields with different meaning separated by single 0x00 byte. [Error]: SecurityErrorDomain::kBruteForceRisk if key length of the sourceKm is below of an internally defined limitation [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible [Error]: SecurityErrorDomain::kUsageViolation if this transformation type is prohibited by the "allowed usage" restrictions of the provided sourceKm or salt objects [Error]: SecurityErrorDomain::kInvalidArgument if any of arguments is incorrect, and the error condition is not covered by the cases described above	

⌋*(RS_CRYPTO_02007, RS_CRYPTO_02113, RS_CRYPTO_02404)*

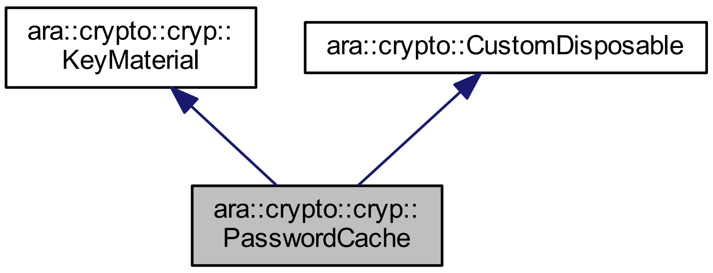
8.18.1.3 class ara::crypto::cryp::PasswordCache

[SWS_CRYPT_22300]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::PasswordCache
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::KeyMaterial
Syntax:	`class PasswordCache : public KeyMaterial {...};`
Header file:	#include "ara/crypto/cryp/password_cache.h"
Description:	Password secure cache context interface

⌋*(RS_CRYPTO_02106)*

Inheritance diagram for ara::crypto::cryp::PasswordCache:



Public Types

- using Uptr = std::unique_ptr< PasswordCache, CustomDeleter >

Public Member Functions

- virtual std::size_t GetMaximalLength () const noexcept=0
- virtual std::size_t GetRequiredLength () const noexcept=0
- virtual unsigned GetRequiredComplexity () const noexcept=0
- virtual void Clear () noexcept=0
- virtual std::size_t GetLength () const noexcept=0
- virtual unsigned GetComplexity () const noexcept=0
- virtual ara::core::Result< void > Reset (ara::core::StringView password) noexcept=0

- virtual bool Compare (ara::core::StringView password) const noexcept=0
- virtual ara::core::Result< PasswordHash::Uptr > SecureHash (HashFunctionCtx &hash, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0
- virtual ara::core::Result< bool > Verify (HashFunctionCtx &hashCtx, const PasswordHash &passwordHash) const noexcept=0
- virtual bool AskUser (ara::core::StringView prompt, bool repeat=false) noexcept=0

Additional Inherited Members

8.18.1.3.1 Member Typedef Documentation

8.18.1.3.1.1 using ara::crypto::cryp::PasswordCache::Uptr = std::unique_ptr<PasswordCache, CustomDeleter>

[SWS_CRYPT_22301]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::PasswordCache::Uptr
Scope:	class ara::crypto::cryp::PasswordCache
Derived from:	std::unique_ptr<PasswordCache, CustomDeleter>
Syntax:	`using ara::crypto::cryp::PasswordCache::Uptr = std::unique_ptr<PasswordCache, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/password_cache.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.18.1.3.2 Member Function Documentation

8.18.1.3.2.1 virtual std::size_t ara::crypto::cryp::PasswordCache::GetMaximalLength ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_22311]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::PasswordCache::GetMaximalLength()
Scope:	class ara::crypto::cryp::PasswordCache
Syntax:	`virtual std::size_t GetMaximalLength () const noexcept=0;`

▽

△

Return value:	std::size_t	maximal supported password length (or buffer size) in characters
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Get maximal password length.	

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.2 virtual std::size_t ara::crypto::cryp::PasswordCache::GetRequiredLength ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22312]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::GetRequiredLength()	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual std::size_t GetRequiredLength () const noexcept=0;`	
Return value:	std::size_t	minimal required password length in characters
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Get required password length.	

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.3 virtual unsigned ara::crypto::cryp::PasswordCache::GetRequiredComplexity ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22313]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::GetRequiredComplexity()	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual unsigned GetRequiredComplexity () const noexcept=0;`	
Return value:	unsigned	minimal required password complexity (0 == no requirements)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	

▽

△

Description:	Get required password complexity.
Notes:	Each symbol category in requirements means +1 to the complexity (f.e.: lower/upper cases, numbers, special symbols).

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.4 virtual void ara::crypto::cryp::PasswordCache::Clear ( ) `[pure virtual], [noexcept]`

[SWS_CRYPT_22314]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::PasswordCache::Clear()
Scope:	class ara::crypto::cryp::PasswordCache
Syntax:	`virtual void Clear () noexcept=0;`
Return value:	None
Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/password_cache.h"
Description:	Securely clear the password cache.

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.5 virtual std::size_t ara::crypto::cryp::PasswordCache::GetLength ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_22315]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::GetLength()	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual std::size_t GetLength () const noexcept=0;`	
Return value:	std::size_t	actual password length in characters
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Get actual password length.	

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.6 virtual unsigned ara::crypto::cryp::PasswordCache::GetComplexity ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_22316]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::GetComplexity()	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual unsigned GetComplexity () const noexcept=0;`	
Return value:	unsigned	actual password complexity level
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Get actual password complexity.	

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.7 virtual ara::core::Result<void> ara::crypto::cryp::PasswordCache::Reset (ara::core::StringView *password*) `[pure virtual], [noexcept]`

[SWS_CRYPT_22317]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::Reset(ara::core::StringView password)	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual ara::core::Result<void> Reset (ara::core::StringView password) noexcept=0;`	
Parameters (in):	password	a new value of the password
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Reset password context to new value.	
Notes:	[Error]: SecurityErrorDomain::kInvalidInputSize if the new password has a length greater than GetMaximalLength() of this password cache instance	

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.8 virtual bool ara::crypto::cryp::PasswordCache::Compare (ara::core::StringView *password*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_22318]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::Compare(ara::core::StringView password)	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual bool Compare (ara::core::StringView password) const noexcept=0;`	
Parameters (in):	password	an external value for comparison
Return value:	bool	true if internally stored and provided passwords are equal
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Compare password context with provided value for equality.	

⌋*(RS_CRYPTO_02311)*

8.18.1.3.2.9 virtual ara::core::Result<PasswordHash::Uptr> ara::crypto::cryp::PasswordCache::SecureHash (HashFunctionCtx & *hash,* ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22319]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::SecureHash(HashFunctionCtx &hash, ReservedObject Index reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual ara::core::Result<PasswordHash::Uptr> SecureHash (HashFunction Ctx &hash, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	hash	a hash-function context that should be used for the digest calculation
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< PasswordHash::Uptr >	unique smart pointer to the created password hash object
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Calculate secure hash of the password randomized by a salt.	
Notes:	[Error]: SecurityErrorDomain::kIncompleteArgState if the hashCtx context is not initialized by required domain parameters [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋(*RS_CRYPTO_02404*)

8.18.1.3.2.10 virtual ara::core::Result<bool> ara::crypto::cryp::PasswordCache::Verify (HashFunctionCtx & *hashCtx*, const PasswordHash & *passwordHash*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22320]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::Verify(HashFunctionCtx &hashCtx, const PasswordHash &passwordHash)	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual ara::core::Result<bool> Verify (HashFunctionCtx &hashCtx, const PasswordHash &passwordHash) const noexcept=0;`	
Parameters (in):	hashCtx	a hash context that should be used for verification
	passwordHash	a password hash object containing a reference value
Return value:	ara::core::Result< bool >	true if a password stored in this context matches to provided hash value
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Verifies a password stored in this context to compliance to the provided hash value.	
Notes:	Before returning from this method the hashCtx context should be cleaned (from intermediate results)! [Error]: SecurityErrorDomain::kIncompatibleArguments if the hash algorithms in the hash-function context hashCtx and in the PasswordHash object differ	

⌋(*RS_CRYPTO_02311*)

8.18.1.3.2.11 virtual bool ara::crypto::cryp::PasswordCache::AskUser (ara::core::StringView *prompt*, bool *repeat* = `false`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22321]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::PasswordCache::AskUser(ara::core::StringView prompt, bool repeat=false)	
Scope:	class ara::crypto::cryp::PasswordCache	
Syntax:	`virtual bool AskUser (ara::core::StringView prompt, bool repeat=false) noexcept=0;`	
Parameters (in):	prompt	a prompt message that should be displayed to user
	repeat	if value of this flag is true then password must be entered twice (for confirmation)
Return value:	bool	true if password was entered by a user and false if it was canceled by the user or if the method is not supported at all
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/password_cache.h"	
Description:	Optional method for asking a password in secure manner by a top-most GUI window.	
Notes:	Implementation of this method must automatically obtain authentic name of the consumer application (from the Execution Manager) and display it in the window title! The password request window must have two buttons: "Ok" and "Cancel". It is recommended to call this method in a dedicated thread due to prevent blocking the main thread.	

⌋*(RS_CRYPTO_02106)*

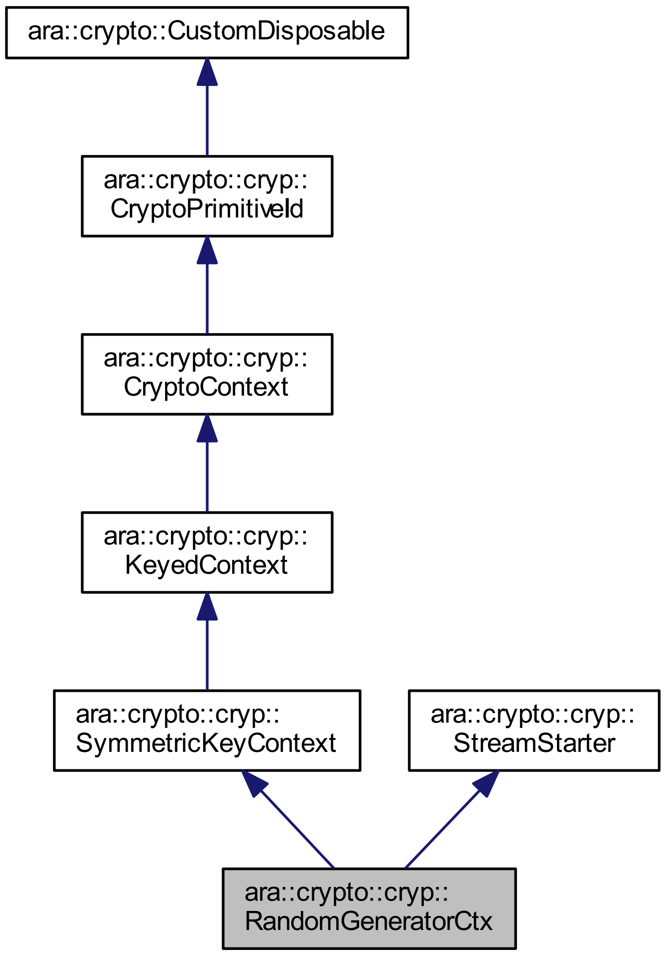
8.18.1.4 class ara::crypto::cryp::RandomGeneratorCtx

[SWS_CRYPT_22900]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::RandomGeneratorCtx
Scope:	namespace ara::crypto::cryp
Base class:	std::enable_shared_from_this< RandomGeneratorCtx >
Syntax:	`class RandomGeneratorCtx : public enable_shared_from_this< RandomGeneratorCtx > {...};`
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"
Description:	Random Number Generator (RNG) Context interface
Notes:	Any user of this interface should create shared pointers to it only by calls of the method shared_from_this()!

⌋*(RS_CRYPTO_02206)*

Inheritance diagram for ara::crypto::cryp::RandomGeneratorCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: SymmetricKeyContext
ara::crypto::cryp:: StreamStarter
ara::crypto::cryp:: RandomGeneratorCtx

Public Types

- using Sptr = std::shared_ptr< RandomGeneratorCtx >

Public Member Functions

- virtual bool AddEntropy (ReadOnlyMemRegion entropy) noexcept=0
- virtual ara::core::Result< void > Generate (WritableMemRegion output) noexcept=0

Additional Inherited Members

8.18.1.4.1 Member Typedef Documentation

8.18.1.4.1.1 using ara::crypto::cryp::RandomGeneratorCtx::Sptr = std::shared_ptr<RandomGeneratorCtx>

[SWS_CRYPT_22901]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::RandomGeneratorCtx::Sptr
Scope:	class ara::crypto::cryp::RandomGeneratorCtx
Derived from:	std::shared_ptr<RandomGeneratorCtx>
Syntax:	`using ara::crypto::cryp::RandomGeneratorCtx::Sptr = std::shared_ptr<RandomGeneratorCtx>;`
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.18.1.4.2 Member Function Documentation

8.18.1.4.2.1 virtual bool ara::crypto::cryp::RandomGeneratorCtx::AddEntropy (ReadOnlyMemRegion *entropy*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_22911]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::RandomGeneratorCtx::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
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Update internal state of the RNG by additional entropy.	
Notes:	This method is optional for implementation! An implementation of this method may "accumulate" provided entropy for future use.	

⌋*(RS_CRYPTO_02311)*

8.18.1.4.2.2 virtual ara::core::Result<void> ara::crypto::cryp::RandomGeneratorCtx::Generate (WritableMemRegion *output*) `[pure virtual], [noexcept]`

[SWS_CRYPT_22913]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::RandomGeneratorCtx::Generate(WritableMemRegion output)	
Scope:	class ara::crypto::cryp::RandomGeneratorCtx	
Syntax:	`virtual ara::core::Result<void> Generate (WritableMemRegion output) noexcept=0;`	
Parameters (out):	output	a target buffer for filling by the generated random sequence
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/random_generator_ctx.h"	
Description:	Fill whole provided buffer by generated random sequence.	
Notes:	[Error]: SecurityErrorDomain::kUninitializedContext if this context implements a deterministic RNG, but it was not initialized by a key value	

⌋*(RS_CRYPTO_02311)*

8.19 Symmetric transformation interfaces

Classes

- class ara::crypto::cryp::AuthnStreamCipherCtx
- class ara::crypto::cryp::KeyDiversifierCtx
- class ara::crypto::cryp::MessageAuthnCodeCtx
- class ara::crypto::cryp::StreamCipherCtx
- class ara::crypto::cryp::SymmetricBlockCipherCtx
- class ara::crypto::cryp::SymmetricKeyWrapperCtx

Detailed Description

This group consists of top-level interfaces of symmetric transformations only.

8.19.1 Class Documentation

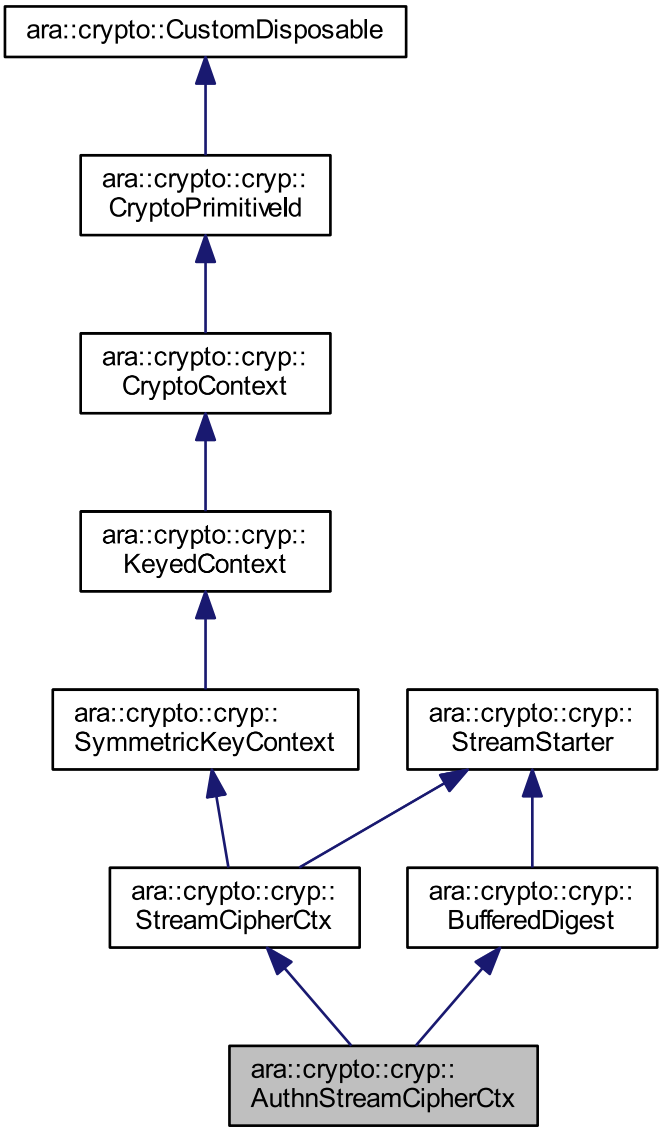
8.19.1.1 class ara::crypto::cryp::AuthnStreamCipherCtx

[SWS_CRYPT_20100]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::AuthnStreamCipherCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::StreamCipherCtx
Syntax:	`class AuthnStreamCipherCtx : public StreamCipherCtx {...};`
Header file:	#include "ara/crypto/cryp/authn_stream_cipher_ctx.h"
Description:	Generalized Authenticated Stream Cipher Context interface.
Notes:	Methods of the derived interface BufferedDigest are used for authentication of associated public data. Methods of the derived interface StreamCipherCtx are used for encryption/decryption and authentication of confidential part of message. The data stream processing must be executed in following order: Call one of the Start() methods.Process all associated public data via calls of Update() methods.Process the confidential part of the message via calls of ProcessBlocks(), ProcessBytes() (and optionally FinishBytes()) methods.Call the Finish() method due to finalize the authentication code calculation (and get it optionally).Copy of the calculated MAC may be extracted (by GetDigest()) or compared internally (by Compare()).

⌋*(RS_CRYPTO_02207)*

Inheritance diagram for ara::crypto::cryp::AuthnStreamCipherCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: SymmetricKeyContext
ara::crypto::cryp:: StreamStarter
ara::crypto::cryp:: StreamCipherCtx
ara::crypto::cryp:: BufferedDigest
ara::crypto::cryp:: AuthnStreamCipherCtx

Public Types

- using Uptr = std::unique_ptr< AuthnStreamCipherCtx, CustomDeleter >

Public Member Functions

- virtual std::uint64_t GetMaxAssociatedDataSize () const noexcept=0

Additional Inherited Members

8.19.1.1.1 Member Typedef Documentation

8.19.1.1.1.1 using ara::crypto::cryp::AuthnStreamCipherCtx::Uptr = std::unique_ptr<AuthnStreamCipherCtx, CustomDeleter>

[SWS_CRYPT_20101]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::AuthnStreamCipherCtx::Uptr
Scope:	class ara::crypto::cryp::AuthnStreamCipherCtx
Derived from:	std::unique_ptr<AuthnStreamCipherCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::AuthnStreamCipherCtx::Uptr = std::unique_ptr<AuthnStreamCipherCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/authn_stream_cipher_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.19.1.1.2 Member Function Documentation

8.19.1.1.2.1 virtual std::uint64_t ara::crypto::cryp::AuthnStreamCipherCtx::GetMaxAssociatedDataSize ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_20110]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::AuthnStreamCipherCtx::GetMaxAssociatedDataSize()	
Scope:	class ara::crypto::cryp::AuthnStreamCipherCtx	
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/authn_stream_cipher_ctx.h"	

▽

△

Description:	Get maximal supported size of associated public data.

⌋*(RS_CRYPTO_02309)*

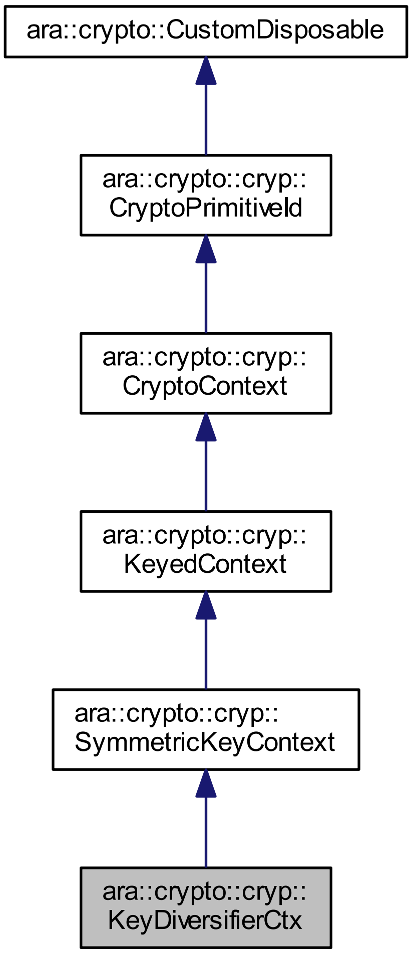
8.19.1.2 class ara::crypto::cryp::KeyDiversifierCtx

[SWS_CRYPT_21600]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyDiversifierCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::SymmetricKeyContext
Syntax:	`class KeyDiversifierCtx : public SymmetricKeyContext {...};`
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"
Description:	Interface of Symmetric Keys Diversification algorithms Context.
Notes:	This interface is dedicated for derivation of multiple "Slave" keys from a single "Master" key, initially loaded to the context. The deversification is executed according to a unique ID of the target "Slave" key.

⌋*(RS_CRYPTO_02103)*

Inheritance diagram for ara::crypto::cryp::KeyDiversifierCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: SymmetricKeyContext
ara::crypto::cryp:: KeyDiversifierCtx

Public Types

- using Uptr = std::unique_ptr< KeyDiversifierCtx, CustomDeleter >

Public Member Functions

- virtual std::size_t GetKeyIdSize () const noexcept=0
- virtual std::size_t GetFillerSize () const noexcept=0
- virtual std::size_t GetTargetKeyBitLength () const noexcept=0

- virtual ara::core::Result< void > Init (ReadOnlyMemRegion appFiller, AlgId targetAlgId=kAlgIdAny, Key::Usage allowedUsage=kAllowPrototypedOnly, DomainParameters::Sptrc params=nullptr) noexcept=0
- virtual ara::core::Result< void > Init (const SecretSeed &appFiller, AlgId targetAlgId=kAlgIdAny, Key::Usage allowedUsage=kAllowPrototypedOnly, DomainParameters::Sptrc params=nullptr) noexcept=0
- virtual AlgId GetTargetAlgId () const noexcept=0
- virtual Key::Usage GetTargetAllowedUsage () const noexcept=0
- virtual ara::core::Result< SymmetricKey::Uptrc > Diversify (ReadOnlyMemRegion targetKeyId, bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

Additional Inherited Members

8.19.1.2.1 Member Typedef Documentation

8.19.1.2.1.1 using ara::crypto::cryp::KeyDiversifierCtx::Uptr = std::unique_ptr<KeyDiversifierCtx, CustomDeleter>

[SWS_CRYPT_21601]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::Uptr
Scope:	class ara::crypto::cryp::KeyDiversifierCtx
Derived from:	std::unique_ptr<KeyDiversifierCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::KeyDiversifierCtx::Uptr = std::unique_ptr<KeyDiversifierCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.19.1.2.2 Member Function Documentation

8.19.1.2.2.1 virtual std::size_t ara::crypto::cryp::KeyDiversifierCtx::GetKeyIdSize ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21611]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::GetKeyIdSize()	
Scope:	class ara::crypto::cryp::KeyDiversifierCtx	
Syntax:	`virtual std::size_t GetKeyIdSize () const noexcept=0;`	
Return value:	std::size_t	size of the key ID in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"	
Description:	Get the fixed size of the target key ID required by diversification algorithm.	
Notes:	Returned value is constant for each instance of the interface, i.e. independent from configuration by the Init() call.	

⌋*(RS_CRYPTO_02311)*

8.19.1.2.2.2 virtual std::size_t ara::crypto::cryp::KeyDiversifierCtx::GetFillerSize ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21612]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::GetFillerSize()	
Scope:	class ara::crypto::cryp::KeyDiversifierCtx	
Syntax:	`virtual std::size_t GetFillerSize () const noexcept=0;`	
Return value:	std::size_t	size of the application specific filler in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"	
Description:	Get the fixed size of an application specific "filler" required by the context initialization.	
Notes:	Returned value is constant for each instance of the interface, i.e. independent from configuration by the Init() call.	

⌋*(RS_CRYPTO_02311)*

8.19.1.2.2.3 virtual std::size_t ara::crypto::cryp::KeyDiversifierCtx::GetTargetKeyBitLength ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21613]{DRAFT} ⌈

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

⌋(*RS_CRYPTO_02311*)

8.19.1.2.2.4 virtual ara::core::Result<void> ara::crypto::cryp::KeyDiversifierCtx::Init (ReadOnlyMemRegion *appFiller*, AlgId *targetAlgId* = kAlgIdAny, Key::Usage *allowedUsage* = kAllowPrototypedOnly, DomainParameters::Sptrc *params* = `nullptr`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21614]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::Init(ReadOnlyMemRegion appFiller, AlgId targetAlgId=kAlgIdAny, Key::Usage allowedUsage=kAllowPrototypedOnly, DomainParameters::Sptrc params=nullptr)	
Scope:	class ara::crypto::cryp::KeyDiversifierCtx	
Syntax:	`virtual ara::core::Result<void> Init (ReadOnlyMemRegion appFiller, AlgId targetAlgId=kAlgIdAny, Key::Usage allowedUsage=kAllowPrototypedOnly, DomainParameters::Sptrc params=nullptr) noexcept=0;`	
Parameters (in):	appFiller	an application specific "filler" value
	targetAlgId	the identifier of target symmetric crypto algorithm
	allowedUsage	bit-flags that define a list of allowed transformations' types in which the target key can be used
	params	an optional pointer to the domain parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"	
Description:	Execute initialization of the diversifier context by setting a public "filler" value.	

▽

△

Notes:	If (GetFillerSize() == 0) then the appFiller argument can have 0 size, because it is ignored in any case. If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the generated key object. If (targetAlgId == kAlgIdAny) then a diversified key can be loaded to any symmetric context that supports same key length (if the "allowed usage" flags are also satisfied)! [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kInvalidInputSize if size of the appFiller is incorrect, i.e. if (app Filler.size() < GetFillerSize()); [Error]: SecurityErrorDomain::kIncompatibleArguments if targetAlgId specifies a cryptographic algorithm different from a symmetric one with key length equal to GetTargetKeyLength(); [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object is in an incompleted state [Error]: SecurityErrorDomain::kUsageViolation if allowedUsage specifies usage of the slave key incompatible with prototyped value of the master key loaded to the context

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

8.19.1.2.2.5 virtual ara::core::Result<void> ara::crypto::cryp::KeyDiversifierCtx::Init (const SecretSeed & *appFiller*, AlgId *targetAlgId* = kAlgIdAny, Key::Usage *allowedUsage* = kAllowPrototypedOnly, DomainParameters::Sptrc *params* = `nullptr`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21615]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::Init(const SecretSeed &appFiller, AlgId targetAlgId=kAlgId Any, Key::Usage allowedUsage=kAllowPrototypedOnly, DomainParameters::Sptrc params=nullptr)	
Scope:	class ara::crypto::cryp::KeyDiversifierCtx	
Syntax:	`virtual ara::core::Result<void> Init (const SecretSeed &appFiller, Alg Id targetAlgId=kAlgIdAny, Key::Usage allowedUsage=kAllowPrototyped Only, DomainParameters::Sptrc params=nullptr) noexcept=0;`	
Parameters (in):	appFiller	the application specific "filler" value
	targetAlgId	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

▽

△

	params	an optional pointer to Domain Parameters required for full specification of the target symmetric transformation (e.g. variable S-boxes of GOST28147-89), and therefore should be associated with the target symmetric key
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"	
Description:	Execute initialization of the diversifier context by setting a secret "filler" value.	
Notes:	If (GetFillerSize() == 0) then the appFiller argument can have 0 size, because it is ignored in any case. If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the generated key object. If (targetAlgId == kAlgIdAny) then a diversified key can be loaded to any symmetric context that supports same key length (if the "allowed usage" flags are also satisfied)! [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kInvalidInputSize if size of the appFiller is incorrect, i.e. if (app Filler.size() < GetFillerSize()); [Error]: SecurityErrorDomain::kIncompatibleArguments if targetAlgId specifies a cryptographic algorithm different from a symmetric one with key length equal to GetTargetKeyLength(); [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kUsageViolation if this transformation type is prohibited by the "allowed usage" restrictions of the provided appFiller object; or if allowedUsage specifies usage of the slave key incompatible with prototyped value of the master key loaded to the content	

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

8.19.1.2.2.6 virtual AlgId ara::crypto::cryp::KeyDiversifierCtx::GetTargetAlgId ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21616]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::GetTargetAlgId()	
Scope:	class ara::crypto::cryp::KeyDiversifierCtx	
Syntax:	`virtual AlgId GetTargetAlgId () const noexcept=0;`	
Return value:	AlgId	the symmetric algorithm ID of target keys, configured by last call of the Init() method

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"
Description:	Get the symmetric algorithm ID of target (slave) keys.
Notes:	If the context was not configured yet by a call of the Init() method then kAlgIdUndefined should be returned.

⌋(*RS_CRYPTO_02311*)

8.19.1.2.2.7 virtual Key::Usage ara::crypto::cryp::KeyDiversifierCtx::GetTargetAllowedUsage ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_21617]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::GetTargetAllowedUsage()	
Scope:	class ara::crypto::cryp::KeyDiversifierCtx	
Syntax:	`virtual Key::Usage GetTargetAllowedUsage () const noexcept=0;`	
Return value:	Key::Usage	allowed key usage bit-flags of target keys
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"	
Description:	Get allowed key usage of target (slave) keys.	
Notes:	Returned value depends from the master key prototype and the argument targetAlgId of last call of the Init() method. If the context was not configured yet by a call of the Init() method then a prototyped value of the master key should be returned.	

⌋(*RS_CRYPTO_02008*)

8.19.1.2.2.8 virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::KeyDiversifierCtx::Diversify (ReadOnlyMemRegion *targetKeyId*, bool *isSession* = `true`, bool *isExportable* = `false`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual], [noexcept]`

[SWS_CRYPT_21618]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDiversifierCtx::Diversify(ReadOnlyMemRegion targetKeyId, bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::KeyDiversifierCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> Diversify (ReadOnlyMemRegion targetKeyId, bool isSession=true, bool isExportable=false, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	targetKeyId	an ID of the target key
	isSession	the "session" (or "temporary") attribute for the target key (if true)
	isExportable	the exportability attribute for the target key (if true)
	reservedIndex	an optional index of reserved object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< SymmetricKey::Uptrc >	unique smart pointer to the diversified slave key
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_diversifier_ctx.h"	
Description:	Execute the key diversification from provided key ID.	
Notes:	[Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value; or if (0 < GetFillerSize()), but the context was not initialized by a "filler" value via call of the Init() method [Error]: SecurityErrorDomain::kInvalidInputSize if size of the targetKeyId are incorrect, e.g. if (targetKeyId.size() < GetKeyIdSize()) [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if size of the slot specified by reservedIndex is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02113, RS_CRYPTO_02404)*

8.19.1.3 class ara::crypto::cryp::MessageAuthnCodeCtx

[SWS_CRYPT_22100]{DRAFT} ⌈

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

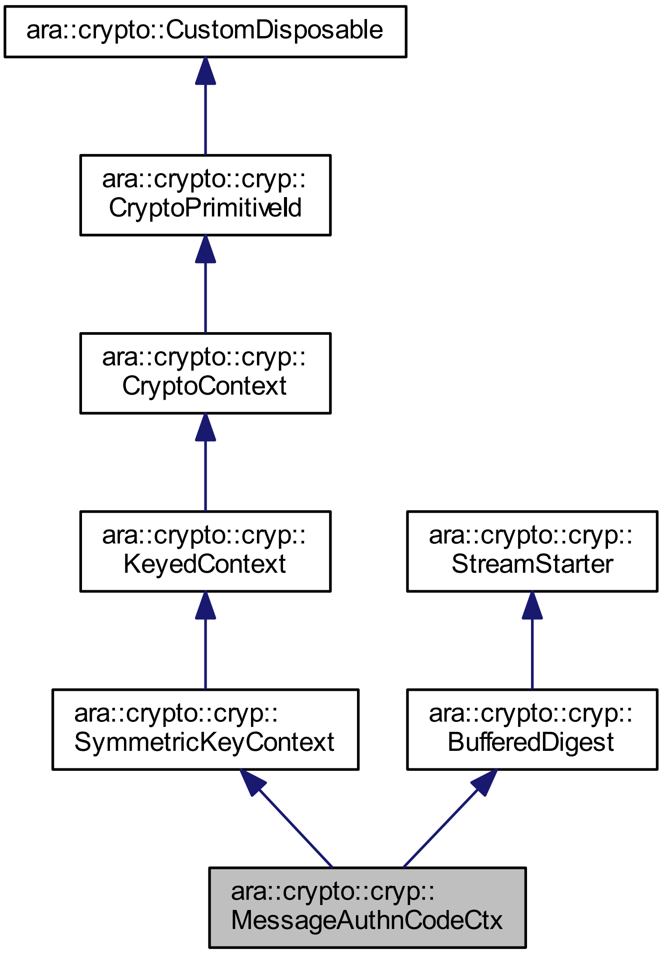
▽

△

Description:	Keyed Message Authentication Code Context interface definition (MAC/HMAC).

⌋*(RS_CRYPTO_02203)*

Inheritance diagram for ara::crypto::cryp::MessageAuthnCodeCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: StreamStarter
ara::crypto::cryp:: SymmetricKeyContext
ara::crypto::cryp:: BufferedDigest
ara::crypto::cryp:: MessageAuthnCodeCtx

Public Types

- using Uptr = std::unique_ptr< MessageAuthnCodeCtx, CustomDeleter >

Additional Inherited Members

8.19.1.3.1 Member Typedef Documentation

8.19.1.3.1.1 using ara::crypto::cryp::MessageAuthnCodeCtx::Uptr = std::unique_ptr<MessageAuthnCodeCtx, CustomDeleter>

[SWS_CRYPT_22101]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::MessageAuthnCodeCtx::Uptr
Scope:	class ara::crypto::cryp::MessageAuthnCodeCtx
Derived from:	std::unique_ptr<MessageAuthnCodeCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::MessageAuthnCodeCtx::Uptr = std::unique_ptr<MessageAuthnCodeCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/message_authn_code_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

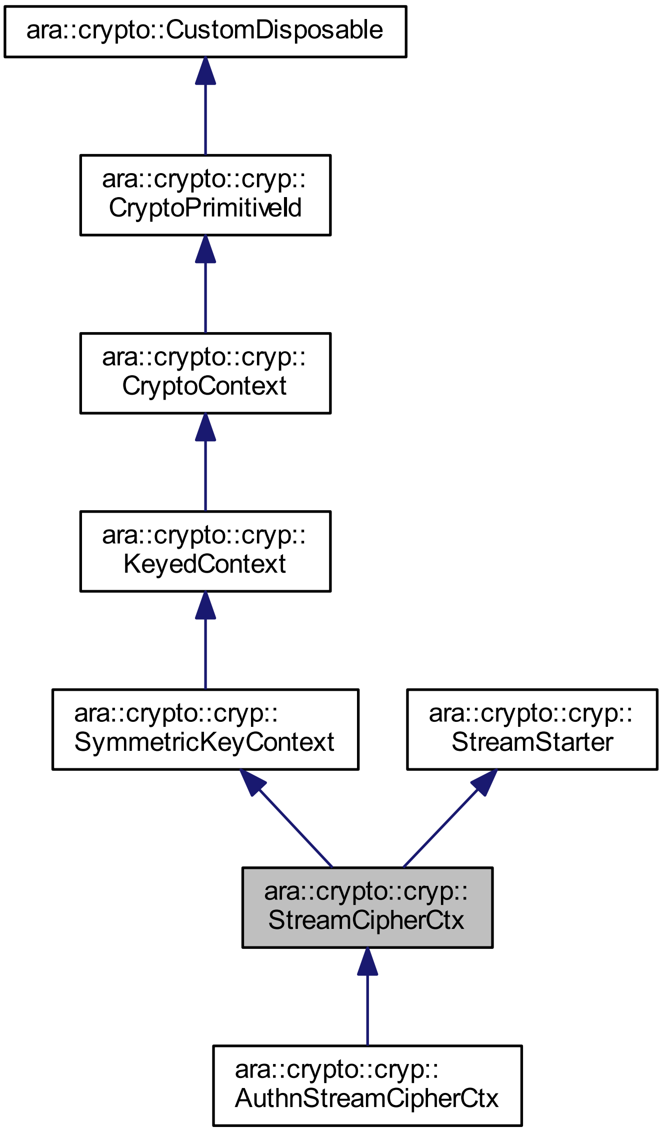
8.19.1.4 class ara::crypto::cryp::StreamCipherCtx

[SWS_CRYPT_23600]{DRAFT} ⌈

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

Inheritance diagram for ara::crypto::cryp::StreamCipherCtx:



Public Types

- using Uptr = std::unique_ptr< StreamCipherCtx, CustomDeleter >

Public Member Functions

- virtual bool IsBytewiseMode () const noexcept=0
- virtual bool IsSeekableMode () const noexcept=0
- virtual ara::core::Result< void > Seek (std::int64_t offset, bool fromBegin=true) noexcept=0
- virtual ara::core::Result< void > ProcessBlocks (WritableMemRegion out, ReadOnlyMemRegion in) noexcept=0
- virtual ara::core::Result< void > ProcessBlocks (ReadWriteMemRegion inOut) noexcept=0
- virtual ara::core::Result< std::size_t > ProcessBytes (WritableMemRegion out, ReadOnlyMemRegion in) noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< void > ProcessBytes (ByteVectorT< Alloc > &out, ReadOnlyMemRegion in) noexcept
- virtual ara::core::Result< std::size_t > FinishBytes (WritableMemRegion out, ReadOnlyMemRegion in) noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< void > FinishBytes (ByteVectorT< Alloc > &out, ReadOnlyMemRegion in) noexcept
- virtual std::size_t CountBytesInCache () const noexcept=0
- std::size_t EstimateMaxInputSize (std::size_t outputCapacity) const noexcept
- std::size_t EstimateRequiredCapacity (std::size_t inputSize, bool isFinal=false) const noexcept

Additional Inherited Members

8.19.1.4.1 Member Typedef Documentation

8.19.1.4.1.1 using ara::crypto::cryp::StreamCipherCtx::Uptr = std::unique_ptr<StreamCipherCtx, CustomDeleter>

[SWS_CRYPT_23601]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::StreamCipherCtx::Uptr
Scope:	class ara::crypto::cryp::StreamCipherCtx
Derived from:	std::unique_ptr<StreamCipherCtx, CustomDeleter>

⌋

△

Syntax:	`using ara::crypto::cryp::StreamCipherCtx::Uptr = std::unique_ptr<StreamCipherCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.19.1.4.2 Member Function Documentation

8.19.1.4.2.1 virtual bool ara::crypto::cryp::StreamCipherCtx::IsBytewiseMode ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_23611]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::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)*

8.19.1.4.2.2 virtual bool ara::crypto::cryp::StreamCipherCtx::IsSeekableMode ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_23612]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::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*)

8.19.1.4.2.3 virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::Seek (std::int64_t *offset,* bool *fromBegin* = `true`) `[pure virtual], [noexcept]`

[SWS_CRYPT_23613]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::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	
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.	
Notes:	[Error]: SecurityErrorDomain::kUnsupported if the seek operation is not supported by the current mode [Error]: SecurityErrorDomain::kProcessingNotStarted if the data processing was not started by a call of the Start() method [Error]: 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) [Error]: SecurityErrorDomain::kInvalidArgument if the offset is not aligned to the required boundary (see IsBytewiseMode())	

⌋(*RS_CRYPTO_02304*)

8.19.1.4.2.4 virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::ProcessBlocks (WritableMemRegion *out,* ReadOnlyMemRegion *in*) `[pure virtual], [noexcept]`

[SWS_CRYPT_23614]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::ProcessBlocks(WritableMemRegion out, ReadOnlyMem Region in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<void> ProcessBlocks (WritableMemRegion out, ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	an output data buffer
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"	
Description:	Processe initial parts of message aligned to the block-size boundary.	
Notes:	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! [Error]: SecurityErrorDomain::kIncompatibleArguments if sizes of the input and output buffers are not equal [Error]: SecurityErrorDomain::kInvalidInputSize if size of the input buffer is not divisible by the block size (see GetBlockSize()) [Error]: SecurityErrorDomain::kInOutBuffersIntersect if the input and output buffers partially intersect [Error]: SecurityErrorDomain::kInvalidUsageOrder if this method is called after processing of non-aligned data (to the block-size boundary) [Error]: SecurityErrorDomain::kProcessingNotStarted if the data processing was not started by a call of the Start() method	

⌋*(RS_CRYPTO_02302)*

8.19.1.4.2.5 virtual ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::ProcessBlocks (ReadWriteMemRegion *inOut*) `[pure virtual], [noexcept]`

[SWS_CRYPT_23615]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::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	

▽

△

Header file:	#include "ara/crypto/cryp/stream_cipher_ctx.h"
Description:	Processe initial parts of message aligned to the block-size boundary.
Notes:	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! [Error]: SecurityErrorDomain::kInvalidInputSize if size of the inOut buffer is not divisible by the block size (see GetBlockSize()) [Error]: SecurityErrorDomain::kInvalidUsageOrder if this method is called after processing of non-aligned data (to the block-size boundary) [Error]: SecurityErrorDomain::kProcessingNotStarted if the data processing was not started by a call of the Start() method

⌋*(RS_CRYPTO_02302)*

8.19.1.4.2.6 virtual ara::core::Result<std::size_t> ara::crypto::cryp::StreamCipherCtx::ProcessBytes (WritableMemRegion *out*, ReadOnlyMemRegion *in*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23616]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::ProcessBytes(WritableMemRegion out, ReadOnlyMem Region in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<std::size_t> ProcessBytes (WritableMemRegion out, ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	an output data buffer
Return value:	ara::core::Result< std::size_t >	actual size of output data (i.e. the number of leading bytes updated in the output buffer)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
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).	
Notes:	If (IsBytewiseMode() == false) then it must be: bs = GetBlockSize(), out.size() >= (((in.size() + bs - 1) / bs) * bs) If (IsBytewiseMode() == true) then it must be: out.size() >= in.size() The input and output buffers must not intersect! This method is "copy inefficient", therefore it should be used only in conditions when an application cannot control the chunking of the original message! [Error]: SecurityErrorDomain::kInsufficientCapacity if the output buffer has capacity insufficient for placing of the transformation result [Error]: SecurityErrorDomain::kInOutBuffersIntersect if the input and output buffers intersect [Error]: SecurityErrorDomain::kProcessingNotStarted if data processing was not started by a call of the Start() method	

⌋*(RS_CRYPTO_02302)*

8.19.1.4.2.7 template<typename Alloc = DefBytesAllocator> ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::ProcessBytes (ByteVectorT< Alloc > & *out*, ReadOnlyMemRegion *in*) `[noexcept]`

[SWS_CRYPT_23617]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::ProcessBytes(ByteVectorT< Alloc > &out, ReadOnlyMemRegion in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`template <typename Alloc>` `inline ara::core::Result<void> ProcessBytes (ByteVectorT< Alloc > &out, ReadOnlyMemRegion in) noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	an input data buffer
Parameters (out):	out	a managed container for the output data
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
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).	
Notes:	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! [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the output container is not enough [Error]: SecurityErrorDomain::kInOutBuffersIntersect if the input buffer points inside of the preallocated output container [Error]: SecurityErrorDomain::kProcessingNotStarted if data processing was not started by a call of the Start() method	

⌋*(RS_CRYPTO_02302)*

8.19.1.4.2.8 virtual ara::core::Result<std::size_t> ara::crypto::cryp::StreamCipherCtx::FinishBytes (WritableMemRegion *out*, ReadOnlyMemRegion *in*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23618]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::FinishBytes(WritableMemRegion out, ReadOnlyMem Region in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`virtual ara::core::Result<std::size_t> FinishBytes (WritableMemRegion out, ReadOnlyMemRegion in) noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	an output data buffer
Return value:	ara::core::Result< std::size_t >	actual size of output data (i.e. the number of leading bytes that were updated in the output buffer)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
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).	
Notes:	If (IsBytewiseMode() == false) then it must be: bs = GetBlockSize(), out.size() >= (((in.size() + bs * (IsDirectTransform() ? 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)! [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the output buffer is not enough [Error]: SecurityErrorDomain::kInOutBuffersIntersect if the input and output buffers intersect [Error]: SecurityErrorDomain::kProcessingNotStarted if data processing was not started by a call of the Start() method	

⌋*(RS_CRYPTO_02302)*

8.19.1.4.2.9 template<typename Alloc = DefBytesAllocator> ara::core::Result<void> ara::crypto::cryp::StreamCipherCtx::FinishBytes (ByteVectorT< Alloc > & *out*, ReadOnlyMemRegion *in*) [noexcept]

[SWS_CRYPT_23619]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::FinishBytes(ByteVectorT< Alloc > &out, ReadOnlyMem Region in)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	

Syntax:	`template <typename Alloc>`
	`inline ara::core::Result<void> FinishBytes (ByteVectorT< Alloc > &out, ReadOnlyMemRegion in) noexcept;`	

Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	in	an input data buffer
Parameters (out):	out	a managed container for output data
Return value:	ara::core::Result< void >	–

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
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).
Notes:	This method sets the size of the output container according to actually saved value. If (IsBytewiseMode() == false) then it must be: bs = GetBlockSize(), out.capacity() >= (((in.size() + bs * (IsDirectTransform() ? 2 : 1) - 1) / bs) * bs) If (IsBytewiseMode() == 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)! The input buffer must not point inside the output container! [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the output container is not enough [Error]: SecurityErrorDomain::kInOutBuffersIntersect if the input and output buffers intersect [Error]: SecurityErrorDomain::kProcessingNotStarted if data processing was not started by a call of the Start() method

⌋*(RS_CRYPTO_02302)*

8.19.1.4.2.10 virtual std::size_t ara::crypto::cryp::StreamCipherCtx::CountBytesInCache ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23620]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::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.	
Notes:	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)*

8.19.1.4.2.11 std::size_t ara::crypto::cryp::StreamCipherCtx::EstimateMaxInputSize (std::size_t *outputCapacity*) const `[noexcept]`

[SWS_CRYPT_23621]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::EstimateMaxInputSize(std::size_t outputCapacity)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`inline 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)*

8.19.1.4.2.12 std::size_t ara::crypto::cryp::StreamCipherCtx::EstimateRequiredCapacity (std::size_t *inputSize*, bool *isFinal* `= false`) const `[noexcept]`

[SWS_CRYPT_23622]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::StreamCipherCtx::EstimateRequiredCapacity(std::size_t inputSize, bool isFinal=false)	
Scope:	class ara::crypto::cryp::StreamCipherCtx	
Syntax:	`inline 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)*

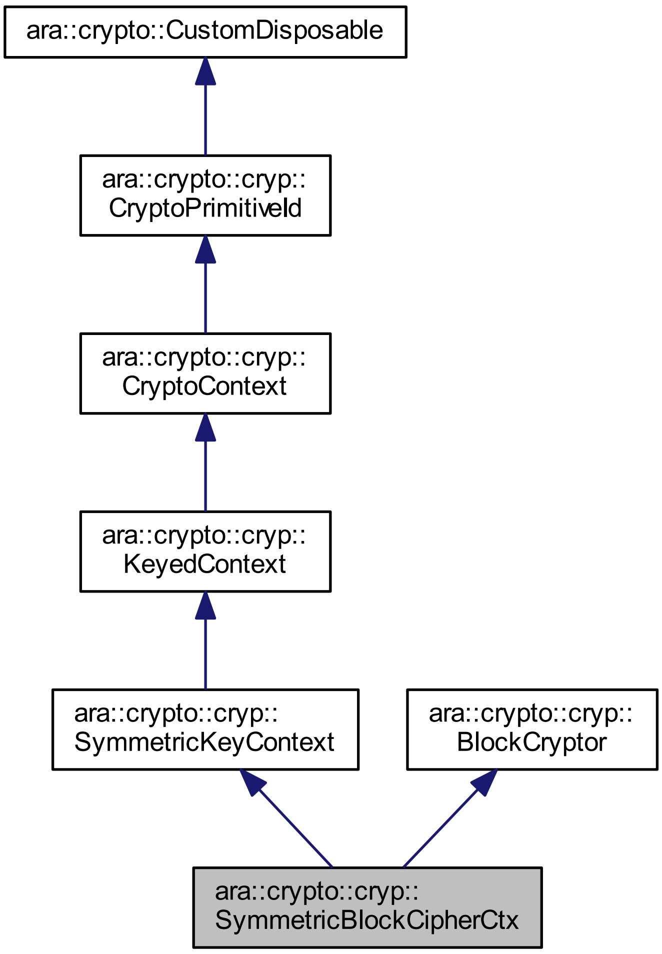
8.19.1.5 class ara::crypto::cryp::SymmetricBlockCipherCtx

[SWS_CRYPT_23700]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02201)*

Inheritance diagram for ara::crypto::cryp::SymmetricBlockCipherCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: SymmetricKeyContext
ara::crypto::cryp:: BlockCryptor
ara::crypto::cryp:: SymmetricBlockCipherCtx

Public Types

- using Uptr = std::unique_ptr< SymmetricBlockCipherCtx, CustomDeleter >

Public Member Functions

- virtual std::size_t GetBlockSize () const noexcept=0
- virtual ara::core::Result< void > ProcessBlocks (WritableMemRegion out, ReadOnlyMemRegion in) const noexcept=0

Additional Inherited Members

8.19.1.5.1 Member Typedef Documentation

8.19.1.5.1.1 using ara::crypto::cryp::SymmetricBlockCipherCtx::Uptr = std::unique_ptr<SymmetricBlockCipherCtx, CustomDeleter>

[SWS_CRYPT_23701]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::SymmetricBlockCipherCtx::Uptr
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx
Derived from:	std::unique_ptr<SymmetricBlockCipherCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::SymmetricBlockCipherCtx::Uptr = std::unique_ptr<SymmetricBlockCipherCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.19.1.5.2 Member Function Documentation

8.19.1.5.2.1 virtual std::size_t ara::crypto::cryp::SymmetricBlockCipherCtx::GetBlockSize ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_23711]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricBlockCipherCtx::GetBlockSize()	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`virtual std::size_t GetBlockSize () const noexcept=0;`	
Return value:	std::size_t	size of the data block in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_block_cipher_ctx.h"	
Description:	Get fixed size of the input/output block of data.	
Notes:	GetBlockSize() == BlockCryptor::GetMaxInputSize(true) == BlockCryptor::GetMaxOutput Size(true)	

⌋*(RS_CRYPTO_02302, RS_CRYPTO_02309)*

8.19.1.5.2.2 virtual ara::core::Result<void> ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlocks (WritableMemRegion *out*, ReadOnlyMemRegion *in*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_23712]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricBlockCipherCtx::ProcessBlocks(WritableMemRegion out, Read OnlyMemRegion in)	
Scope:	class ara::crypto::cryp::SymmetricBlockCipherCtx	
Syntax:	`virtual ara::core::Result<void> ProcessBlocks (WritableMemRegion out, ReadOnlyMemRegion in) const noexcept=0;`	
Parameters (in):	in	an input data buffer
Parameters (out):	out	an output data buffer
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:	Processe provided blocks without padding.	
Notes:	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! [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kInvalidInputSize if size of the input buffer is not divisible by the block size (see GetBlockSize()) [Error]: SecurityErrorDomain::kIncompatibleArguments if sizes of the input and output buffer are not equal [Error]: SecurityErrorDomain::kInOutBuffersIntersect if the input and output buffers partially intersect	

⌋*(RS_CRYPTO_02302)*

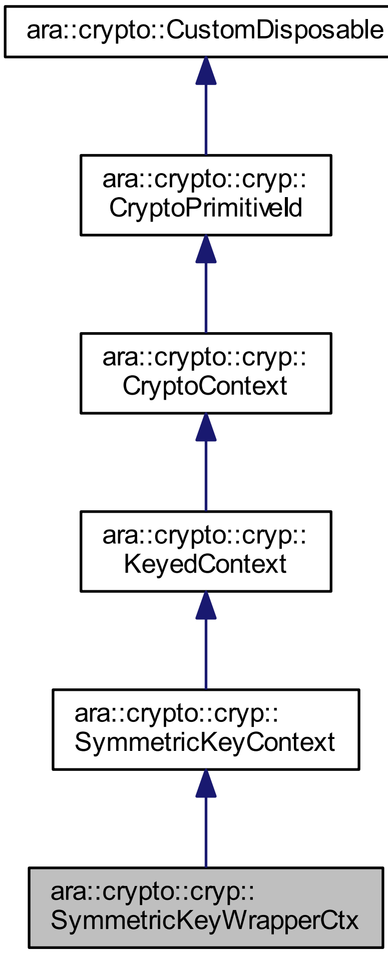
8.19.1.6 class ara::crypto::cryp::SymmetricKeyWrapperCtx

[SWS_CRYPT_24000]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::SymmetricKeyContext
Syntax:	`class SymmetricKeyWrapperCtx : public SymmetricKeyContext {...};`
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).
Notes:	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, RS_CRYPTO_02404)*

Inheritance diagram for ara::crypto::cryp::SymmetricKeyWrapperCtx:



Public Types

- using Uptr = std::unique_ptr< SymmetricKeyWrapperCtx, CustomDeleter >

Public Member Functions

- virtual std::size_t GetTargetKeyGranularity () const noexcept=0
- virtual std::size_t GetMaxTargetKeyLength () const noexcept=0
- virtual std::size_t CalculateWrappedKeySize (std::size_t keyLength) const noexcept=0

- virtual ara::core::Result< void > WrapKeyMaterial (WritableMemRegion wrapped, const KeyMaterial &key) const noexcept=0
- virtual ara::core::Result< SecretSeed::Uptrc > UnwrapSeed (ReadOnlyMemRegion wrappedSeed, AlgId targetAlgId, SecretSeed::Usage allowedUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0
- virtual ara::core::Result< Key::Uptrc > UnwrapKey (ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0
- template<typename ExpectedKey >
  ara::core::Result< typename ExpectedKey::Uptrc > UnwrapConcreteKey (ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) noexcept

Additional Inherited Members

8.19.1.6.1 Member Typedef Documentation

8.19.1.6.1.1 using ara::crypto::cryp::SymmetricKeyWrapperCtx::Uptr = std::unique_ptr<SymmetricKeyWrapperCtx, CustomDeleter>

[SWS_CRYPT_24001]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::Uptr
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx
Derived from:	std::unique_ptr<SymmetricKeyWrapperCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::SymmetricKeyWrapperCtx::Uptr = std::unique_ptr<SymmetricKeyWrapperCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.19.1.6.2 Member Function Documentation

8.19.1.6.2.1 virtual std::size_t ara::crypto::cryp::SymmetricKeyWrapperCtx::GetTargetKeyGranularity ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24011]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::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).	
Notes:	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_02311)*

8.19.1.6.2.2 virtual std::size_t ara::crypto::cryp::SymmetricKeyWrapperCtx::GetMaxTargetKeyLength ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24012]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::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.	
Notes:	This method can be useful for some implementations different from RFC3394 / RFC5649.	

⌋*(RS_CRYPTO_02311)*

8.19.1.6.2.3 virtual std::size_t ara::crypto::cryp::SymmetricKeyWrapperCtx::CalculateWrappedKeySize (std::size_t *keyLength*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_24013]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::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.	
Notes:	This method can be useful for some implementations different from RFC3394 / RFC5649.	

⌋*(RS_CRYPTO_02311)*

8.19.1.6.2.4 virtual ara::core::Result<void> ara::crypto::cryp::SymmetricKeyWrapperCtx::WrapKeyMaterial (WritableMemRegion *wrapped*, const KeyMaterial & *key*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24014]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::WrapKeyMaterial(WritableMemRegion wrapped, const KeyMaterial &key)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<void> WrapKeyMaterial (WritableMemRegion wrapped, const KeyMaterial &key) const noexcept=0;`	
Parameters (in):	key	a key that should be wrapped
Parameters (out):	wrapped	an output buffer for the wrapped key
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:	Execute the "key wrap" operation for the provided key material.	
Notes:	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. [Error]: SecurityErrorDomain::kInsufficientCapacity if the size of the wrapped buffer is not enough for storing the result [Error]: SecurityErrorDomain::kInvalidInputSize if the key object has an unsupported length [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value	

⌋*(RS_CRYPTO_02311)*

8.19.1.6.2.5 virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapSeed (ReadOnlyMemRegion *wrappedSeed,* AlgId *targetAlgId,* SecretSeed::Usage *allowedUsage,* ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24015]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapSeed(ReadOnlyMemRegion wrapped Seed, AlgId targetAlgId, SecretSeed::Usage allowedUsage, ReservedObjectIndex reserved Index=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> UnwrapSeed (ReadOnlyMem Region wrappedSeed, AlgId targetAlgId, SecretSeed::Usage allowedUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) 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
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap.
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	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Execute the "key unwrap" operation for provided BLOB and produce SecretSeed object.	
Notes:	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). [Error]: SecurityErrorDomain::kInvalidInputSize if the size of provided wrapped seed is unsupported [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02007, RS_CRYPTO_02404)*

8.19.1.6.2.6 virtual ara::core::Result<Key::Uptrc> ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapKey (ReadOnlyMemRegion *wrappedKey,* AlgId *algId,* AllowedUsageFlags *allowedUsage,* DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24016]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapKey(ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`virtual ara::core::Result<Key::Uptrc> UnwrapKey (ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) 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
	params	an optional pointer to domain parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Key::Uptrc >	unique smart pointer to Key object, which keeps unwrapped key material
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/symmetric_key_wrapper_ctx.h"	
Description:	Execute the "key unwrap" operation for provided BLOB and produce Key object.	
Notes:	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)! If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the produced key object. SymmetricKey may be unwrapped in following way: SymmetricKey::Uptrc key = SymmetricKey::Cast(UnwrapKey(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! [Error]: SecurityErrorDomain::kInvalidInputSize if the size of provided wrapped key is unsupported [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋(*RS_CRYPTO_02404*, *RS_CRYPTO_02115*)

8.19.1.6.2.7 template<typename ExpectedKey > ara::core::Result<typename ExpectedKey::Uptrc> ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapConcreteKey (ReadOnlyMemRegion *wrappedKey,* AlgId *algId,* AllowedUsageFlags *allowedUsage,* DomainParameters::Sptrc *params* = `nullptr`*,* ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) `[noexcept]`

[SWS_CRYPT_24017]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SymmetricKeyWrapperCtx::UnwrapConcreteKey(ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::SymmetricKeyWrapperCtx	
Syntax:	`template <typename ExpectedKey> inline ara::core::Result<typename ExpectedKey::Uptrc> UnwrapConcreteKey (ReadOnlyMemRegion wrappedKey, AlgId algId, AllowedUsageFlags allowedUsage, DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) 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
	params	an optional pointer to domain parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap

▽

△

Return value:	ara::core::Result< typename ExpectedKey::Uptrc >	unique smart pointer to ExpectedKey object, which keeps unwrapped key material
Exception Safety:	noexcept	
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.	
Notes:	For additional details see UnwrapKey() [Error]: SecurityErrorDomain::kInvalidInputSize if the size of provided wrapped key is unsupported [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02404, RS_CRYPTO_02115)*

8.20 Asymmetric transformation interfaces

Classes

- class ara::crypto::cryp::DecryptorPrivateCtx
- class ara::crypto::cryp::EncryptorPublicCtx
- class ara::crypto::cryp::KeyAgreementPrivateCtx
- class ara::crypto::cryp::KeyDecapsulatorPrivateCtx
- class ara::crypto::cryp::KeyEncapsulatorPublicCtx
- class ara::crypto::cryp::MsgRecoveryPublicCtx
- class ara::crypto::cryp::SigEncodePrivateCtx
- class ara::crypto::cryp::SignerPrivateCtx
- class ara::crypto::cryp::VerifierPublicCtx
- class ara::crypto::cryp::X509RequestSignerCtx

Detailed Description

This group consists of top-level interfaces of asymmetric transformations only.

8.20.1 Class Documentation

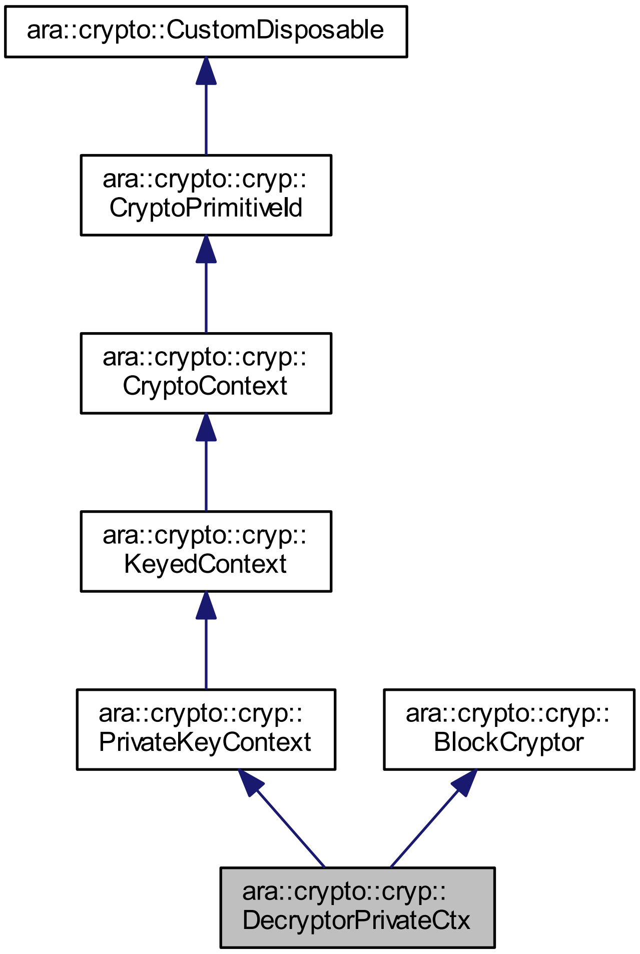
8.20.1.1 class ara::crypto::cryp::DecryptorPrivateCtx

[SWS_CRYPT_20800]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02202)*

Inheritance diagram for ara::crypto::cryp::DecryptorPrivateCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: PrivateKeyContext
ara::crypto::cryp:: BlockCryptor
ara::crypto::cryp:: DecryptorPrivateCtx

Public Types

- using Uptr = std::unique_ptr< DecryptorPrivateCtx, CustomDeleter >

Additional Inherited Members

8.20.1.1.1 Member Typedef Documentation

8.20.1.1.1.1 using ara::crypto::cryp::DecryptorPrivateCtx::Uptr = std::unique_ptr<DecryptorPrivateCtx, CustomDeleter>

[SWS_CRYPT_20801]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::DecryptorPrivateCtx::Uptr
Scope:	class ara::crypto::cryp::DecryptorPrivateCtx
Derived from:	std::unique_ptr<DecryptorPrivateCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::DecryptorPrivateCtx::Uptr = std::unique_ptr<DecryptorPrivateCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/decryptor_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

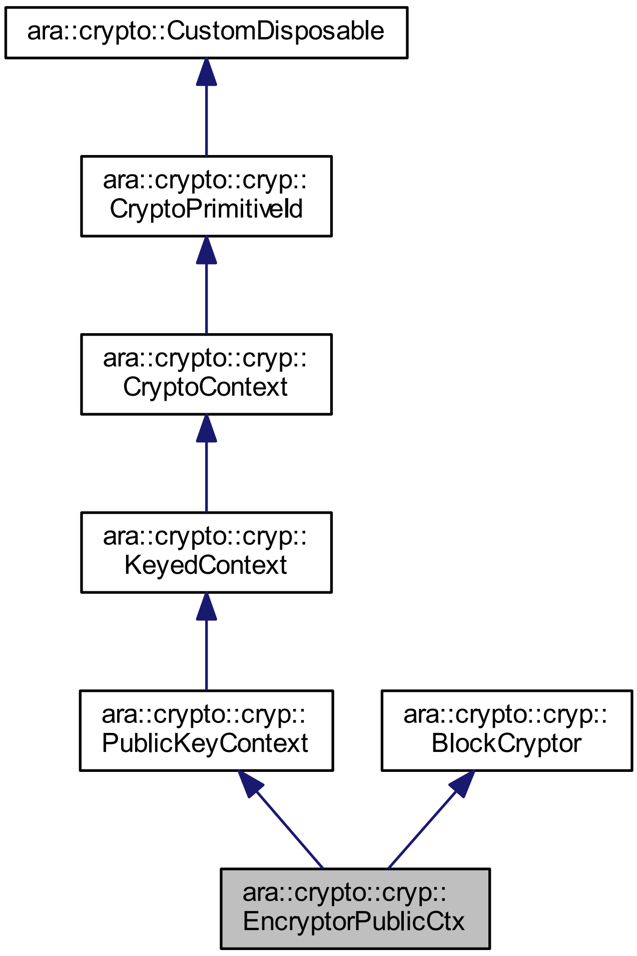
8.20.1.2 class ara::crypto::cryp::EncryptorPublicCtx

[SWS_CRYPT_21000]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02202)*

Inheritance diagram for ara::crypto::cryp::EncryptorPublicCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: PublicKeyContext
ara::crypto::cryp:: BlockCryptor
ara::crypto::cryp:: EncryptorPublicCtx

Public Types

- using Uptr = std::unique_ptr< EncryptorPublicCtx, CustomDeleter >

Additional Inherited Members

8.20.1.2.1 Member Typedef Documentation

8.20.1.2.1.1 using ara::crypto::cryp::EncryptorPublicCtx::Uptr = std::unique_ptr<EncryptorPublicCtx, CustomDeleter>

[SWS_CRYPT_21001]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::EncryptorPublicCtx::Uptr
Scope:	class ara::crypto::cryp::EncryptorPublicCtx
Derived from:	std::unique_ptr<EncryptorPublicCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::EncryptorPublicCtx::Uptr = std::unique_ptr<EncryptorPublicCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/encryptor_public_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

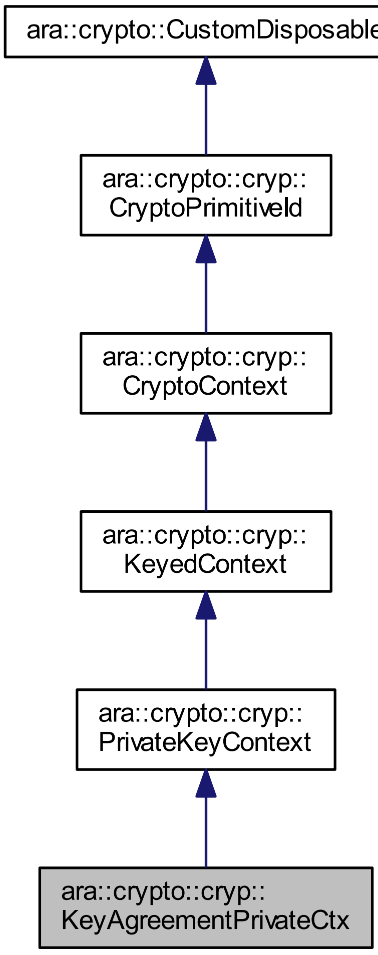
8.20.1.3 class ara::crypto::cryp::KeyAgreementPrivateCtx

[SWS_CRYPT_21300]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyAgreementPrivateCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::PrivateKeyContext
Syntax:	`class KeyAgreementPrivateCtx : public PrivateKeyContext {...};`
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)*

Inheritance diagram for ara::crypto::cryp::KeyAgreementPrivateCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: PrivateKeyContext
ara::crypto::cryp:: KeyAgreementPrivateCtx

Public Types

- using Uptr = std::unique_ptr< KeyAgreementPrivateCtx, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< SecretSeed::Uptrc > AgreeSeed (const PublicKey &otherSideKey, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

- virtual ara::core::Result< SymmetricKey::Uptrc > AgreeKey (const PublicKey &otherSideKey, KeyDerivationFunctionCtx &kdf, AlgId targetAlgId, Key::Usage allowedUsage, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

Additional Inherited Members

8.20.1.3.1 Member Typedef Documentation

8.20.1.3.1.1 using ara::crypto::cryp::KeyAgreementPrivateCtx::Uptr = std::unique_ptr<KeyAgreementPrivateCtx, CustomDeleter>

[SWS_CRYPT_21301]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::KeyAgreementPrivateCtx::Uptr
Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx
Derived from:	std::unique_ptr<KeyAgreementPrivateCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::KeyAgreementPrivateCtx::Uptr = std::unique_ptr<KeyAgreementPrivateCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/key_agreement_private_ctx.h"
Description:	Unique smart pointer of this interface.

⌋*(RS_CRYPTO_02404)*

8.20.1.3.2 Member Function Documentation

8.20.1.3.2.1 virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeSeed (const PublicKey & *otherSideKey*, SecretSeed::Usage *allowedUsage* = kAllowKdfMaterialAnyUsage, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21311]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeSeed(const PublicKey &otherSideKey, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)

▽

△

Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> AgreeSeed (const PublicKey &otherSideKey, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) 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
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
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	
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.	
Notes:	Produced SecretSeed object has following attributes: session, non-exportable, AlgID (this Key-Agreement Algorithm ID). [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kIncompatibleObject if the public and private keys correspond to different algorithms or reference to different domain parameters [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02007, RS_CRYPTO_02404)*

8.20.1.3.2.2 virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeKey (const PublicKey & *otherSideKey,* KeyDerivationFunctionCtx & *kdf,* AlgId *targetAlgId,* Key::Usage *allowedUsage,* ReadOnlyMemRegion *salt* = ReadOnlyMemRegion `()`, ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion `()`, DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21312]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyAgreementPrivateCtx::AgreeKey(const PublicKey &otherSideKey, Key DerivationFunctionCtx &kdf, AlgId targetAlgId, Key::Usage allowedUsage, ReadOnlyMem Region salt=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::KeyAgreementPrivateCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> AgreeKey (const Public Key &otherSideKey, KeyDerivationFunctionCtx &kdf, AlgId targetAlgId, Key::Usage allowedUsage, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), Domain Parameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=k AllocObjectOnHeap) 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)
	params	an optional pointer to Domain Parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< Symmetric Key::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	
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.	
Notes:	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. [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kIncompatibleObject if the public and private keys correspond to different algorithms or reference to different domain parameters or if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object is in an incompleted state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋(*RS_CRYPTO_02115*, *RS_CRYPTO_02404*)

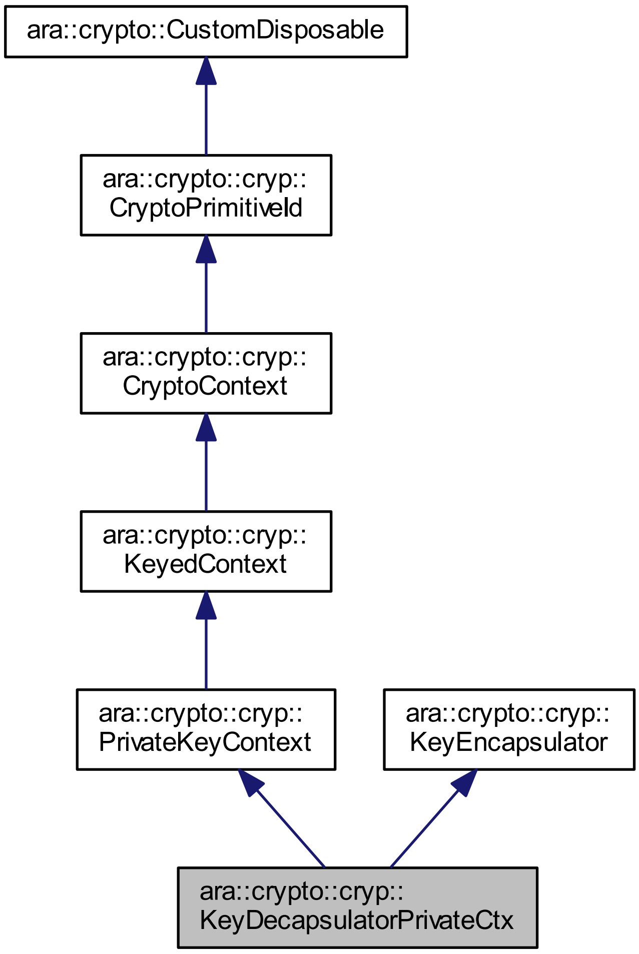
8.20.1.4 class ara::crypto::cryp::KeyDecapsulatorPrivateCtx

[SWS_CRYPT_21400]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyDecapsulatorPrivateCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::PrivateKeyContext
Syntax:	`class KeyDecapsulatorPrivateCtx :  public PrivateKeyContext {...};`
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*, *RS_CRYPTO_02404*)

Inheritance diagram for ara::crypto::cryp::KeyDecapsulatorPrivateCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: PrivateKeyContext
ara::crypto::cryp:: KeyEncapsulator
ara::crypto::cryp:: KeyDecapsulatorPrivateCtx

Public Types

- using Uptr = std::unique_ptr< KeyDecapsulatorPrivateCtx, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< SecretSeed::Uptrc > DecapsulateSeed (ReadOnlyMemRegion input, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

- virtual ara::core::Result< SymmetricKey::Uptrc > DecapsulateKey (ReadOnlyMemRegion input, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

Additional Inherited Members

8.20.1.4.1 Member Typedef Documentation

8.20.1.4.1.1 using ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Uptr = std::unique_ptr<KeyDecapsulatorPrivateCtx, CustomDeleter>

[SWS_CRYPT_21401]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Uptr
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx
Derived from:	std::unique_ptr<KeyDecapsulatorPrivateCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::KeyDecapsulatorPrivateCtx::Uptr = std::unique_ptr<KeyDecapsulatorPrivateCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.20.1.4.2 Member Function Documentation

8.20.1.4.2.1 virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateSeed (ReadOnlyMemRegion *input*, SecretSeed::Usage *allowedUsage* = kAllowKdfMaterialAnyUsage, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21411]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateSeed(ReadOnlyMemRegion input, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)
Scope:	class ara::crypto::cryp::KeyDecapsulatorPrivateCtx

▽

△

Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> DecapsulateSeed (ReadOnlyMemRegion input, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) 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
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
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	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Description:	Decapsulate key material.	
Notes:	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. [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a private key value [Error]: SecurityErrorDomain::kInsufficientCapacity if the output.size() is not enough to save the decapsulation result [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋(*RS_CRYPTO_02007, RS_CRYPTO_02404*)

8.20.1.4.2.2 virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateKey (ReadOnlyMemRegion *input,* KeyDerivationFunctionCtx & *kdf,* AlgId *kekAlgId,* ReadOnlyMemRegion *salt* = ReadOnlyMemRegion `()`, ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion `()`, DomainParameters::Sptrc *params* = `nullptr`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21412]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyDecapsulatorPrivateCtx::DecapsulateKey(ReadOnlyMemRegion input, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMem Region()	
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(), DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) 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)
	params	an optional pointer to domain parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
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	
Header file:	#include "ara/crypto/cryp/key_decapsulator_private_ctx.h"	
Description:	Decapsulate Key Encryption Key (KEK).	
Notes:	Produced SymmetricKey object has following attributes: session, non-exportable, Key Usage: k AllowKeyImporting. If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the generated key object. This method can be used for direct production of the target key, without creation of the intermediate SecretSeed object. [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a private key value [Error]: SecurityErrorDomain::kUnknownIdentifier if kekAlgId specifies incorrect algorithm [Error]: SecurityErrorDomain::kInvalidInputSize if (input.size() <> this->GetEncapsulatedSize()) [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but provided domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but provided domain parameters object has an incomplete state ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115, RS_CRYPTO_02404)*

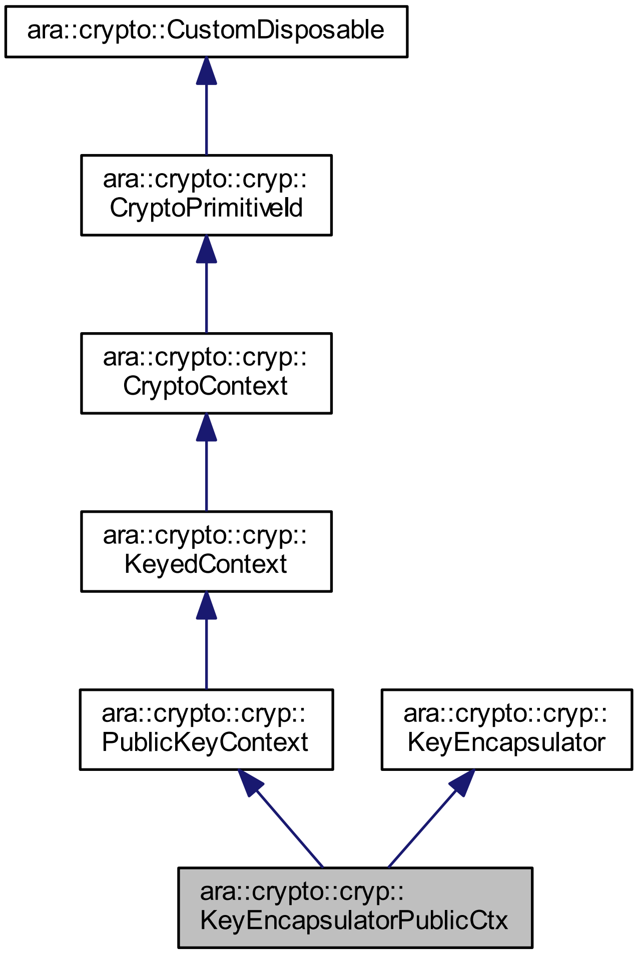
8.20.1.5 class ara::crypto::cryp::KeyEncapsulatorPublicCtx

[SWS_CRYPT_21800]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::KeyEncapsulatorPublicCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::PublicKeyContext
Syntax:	`class KeyEncapsulatorPublicCtx : public PublicKeyContext {...};`
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, RS_CRYPTO_02404)*

Inheritance diagram for ara::crypto::cryp::KeyEncapsulatorPublicCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::KeyedContext
ara::crypto::cryp::PublicKeyContext
ara::crypto::cryp::KeyEncapsulator
ara::crypto::cryp::KeyEncapsulatorPublicCtx

Public Types

- using Uptr = std::unique_ptr< KeyEncapsulatorPublicCtx, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< SecretSeed::Uptrc > EncapsulateSeed (WritableMemRegion output, std::size_t &outSize, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< SecretSeed::Uptrc > EncapsulateSeed (ByteVectorT< Alloc > &output, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept
- virtual ara::core::Result< SymmetricKey::Uptrc > EncapsulateKey (WritableMemRegion output, std::size_t &outSize, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< SymmetricKey::Uptrc > EncapsulateKey (ByteVectorT< Alloc > &output, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept

Additional Inherited Members

8.20.1.5.1 Member Typedef Documentation

8.20.1.5.1.1 using ara::crypto::cryp::KeyEncapsulatorPublicCtx::Uptr = std::unique_ptr<KeyEncapsulatorPublicCtx, CustomDeleter>

[SWS_CRYPT_21801]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::KeyEncapsulatorPublicCtx::Uptr
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx
Derived from:	std::unique_ptr<KeyEncapsulatorPublicCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::KeyEncapsulatorPublicCtx::Uptr = std::unique_ptr<KeyEncapsulatorPublicCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.20.1.5.2 Member Function Documentation

8.20.1.5.2.1 virtual ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateSeed (WritableMemRegion *output*, std::size_t & *outSize*, SecretSeed::Usage *allowedUsage* = kAllowKdfMaterialAnyUsage, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_21811]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateSeed(WritableMemRegion output, std::size_t &outSize, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`virtual ara::core::Result<SecretSeed::Uptrc> EncapsulateSeed (WritableMemRegion output, std::size_t &outSize, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	allowedUsage	the allowed usage scope of the target seed
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Parameters (out):	output	an output buffer (its size should be at least GetEncapsulatedSize() bytes)
	outSize	a variable for getting the actual size of output data (encapsulated key) in bytes
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to SecretSeed object, which keeps the randomly renerated key material encapsulated to the output buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Encapsulate key material (secret seed).	
Notes:	Returned key material should be used for derivation of a symmetric key. Only first GetEncapsulatedSize() bytes of the output buffer can be updated by this method. Produced SecretSeed object has following attributes: session, non-exportable, AlgID = this KEM AlgID. [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a public key value [Error]: SecurityErrorDomain::kInsufficientCapacity if the output.size() is not enough to save the encapsulation result [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02007, RS_CRYPTO_02404)*

8.20.1.5.2.2 template<typename Alloc = DefBytesAllocator> ara::core::Result<SecretSeed::Uptrc> ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateSeed (ByteVectorT< Alloc > & *output*, SecretSeed::Usage *allowedUsage* = kAllowKdfMaterialAnyUsage, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const [noexcept]

[SWS_CRYPT_21812]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateSeed(ByteVectorT< Alloc > &output, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap)	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`template <typename Alloc>` `inline ara::core::Result<SecretSeed::Uptrc> EncapsulateSeed (ByteVectorT< Alloc > &output, SecretSeed::Usage allowedUsage=kAllowKdfMaterialAnyUsage, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
Parameters (in):	allowedUsage	allowed usage scope of the target seed
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Parameters (out):	output	a managed container for output data, i.e. for the "encapsulated key" (its capacity should be at least GetEncapsulatedSize() bytes)
Return value:	ara::core::Result< SecretSeed::Uptrc >	unique smart pointer to SecretSeed object, which keeps the randomly renerated key material encapsulated to the output buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Encapsulate key material (secret seed).	
Notes:	Returned key material should be used for derivation of a symmetric key. This method sets the size of the output container according to actually saved value. Produced SecretSeed object has following attributes: session, non-exportable, AlgID = this KEM AlgID. [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a public key value [Error]: SecurityErrorDomain::kInsufficientCapacity if the output.size() is not enough to save the encapsulation result [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02007, RS_CRYPTO_02404)*

8.20.1.5.2.3 virtual ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateKey (WritableMemRegion *output*, std::size_t & *outSize*, KeyDerivationFunctionCtx & *kdf*, AlgId *kekAlgId*, ReadOnlyMemRegion *salt* = ReadOnlyMemRegion(), ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion(), DomainParameters::Sptrc *params* = nullptr, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const [pure virtual], [noexcept]

[SWS_CRYPT_21813]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateKey(WritableMemRegion output, std::size_t &outSize, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`virtual ara::core::Result<SymmetricKey::Uptrc> EncapsulateKey (WritableMemRegion output, std::size_t &outSize, KeyDerivationFunction Ctx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctxLabel=ReadOnlyMemRegion(), Domain Parameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=k AllocObjectOnHeap) 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)
	params	an optional pointer to domain parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Parameters (out):	output	an output buffer (its size should be at least Get EncapsulatedSize() bytes)
	outSize	a variable for getting the actual size of output data (encapsulated key) in bytes
Return value:	ara::core::Result< Symmetric Key::Uptrc >	unique smart pointer to a symmetric key object derived from a randomly renerated material encapsulated to the output buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Encapsulate Key Encryption Key (KEK).	

⌋

△

Notes:	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. If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the produced key object. This method can be used for direct production of the target key, without creation of the intermediate SecretSeed object. [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a public key value [Error]: SecurityErrorDomain::kInvalidArgument if kekAlgId specifies incorrect algorithm [Error]: SecurityErrorDomain::kInsufficientCapacity if the output.size() is not enough to save the encapsulation result [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but the domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but the domain parameters object has incomplete state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115, RS_CRYPTO_02404)*

8.20.1.5.2.4 template<typename Alloc = DefBytesAllocator> ara::core::Result<SymmetricKey::Uptrc> ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateKey (ByteVectorT< Alloc > & *output*, KeyDerivationFunctionCtx & *kdf*, AlgId *kekAlgId*, ReadOnlyMemRegion *salt* = ReadOnlyMemRegion(), ReadOnlyMemRegion *ctxLabel* = ReadOnlyMemRegion(), DomainParameters::Sptrc *params* = nullptr, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const [noexcept]

[SWS_CRYPT_21814]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::KeyEncapsulatorPublicCtx::EncapsulateKey(ByteVectorT< Alloc > &output, KeyDerivationFunctionCtx &kdf, AlgId kekAlgId, ReadOnlyMemRegion salt=ReadOnlyMem Region()	
Scope:	class ara::crypto::cryp::KeyEncapsulatorPublicCtx	
Syntax:	`template <typename Alloc> inline ara::core::Result<SymmetricKey::Uptrc> EncapsulateKey (Byte VectorT< Alloc > &output, KeyDerivationFunctionCtx &kdf, AlgId kekAlg Id, ReadOnlyMemRegion salt=ReadOnlyMemRegion(), ReadOnlyMemRegion ctx Label=ReadOnlyMemRegion(), DomainParameters::Sptrc params=nullptr, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept;`	
Template param:	Alloc	a custom allocator type of the output container
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)
	params	an optional pointer to domain parameters required for full specification of the symmetric transformation (e.g. variable S-boxes of GOST28147-89)
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Parameters (out):	output	a managed container for the output data, i.e. for the "encapsulated key" (its capacity should be at least GetEncapsulatedSize() bytes)
Return value:	ara::core::Result< Symmetric Key::Uptrc >	unique smart pointer to a symmetric key object derived from randomly renerated material encapsulated to the output buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/key_encapsulator_public_ctx.h"	
Description:	Encapsulate Key Encryption Key (KEK).	
Notes:	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. If (params != nullptr) then the domain parameters object must be in the completed state (see DomainParameters)! If (params != nullptr) then at least the parameters' COUID must be saved to the dependency field of the produced key object. This method can be used for direct production of the target key, without creation of the intermediate SecretSeed object. [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a public key value [Error]: SecurityErrorDomain::kInvalidArgument if kekAlgId specifies incorrect algorithm [Error]: SecurityErrorDomain::kInsufficientCapacity if the output.size() is not enough to save the encapsulation result [Error]: SecurityErrorDomain::kEmptyContainer if the domain parameters are required, but (params == nullptr) was passed ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kIncompatibleObject if (params != nullptr), but the domain parameters object has inappropriate type [Error]: SecurityErrorDomain::kIncompleteArgState if (params != nullptr), but the domain parameters object has incomplete state [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible

⌋*(RS_CRYPTO_02102, RS_CRYPTO_02108, RS_CRYPTO_02115, RS_CRYPTO_02404)*

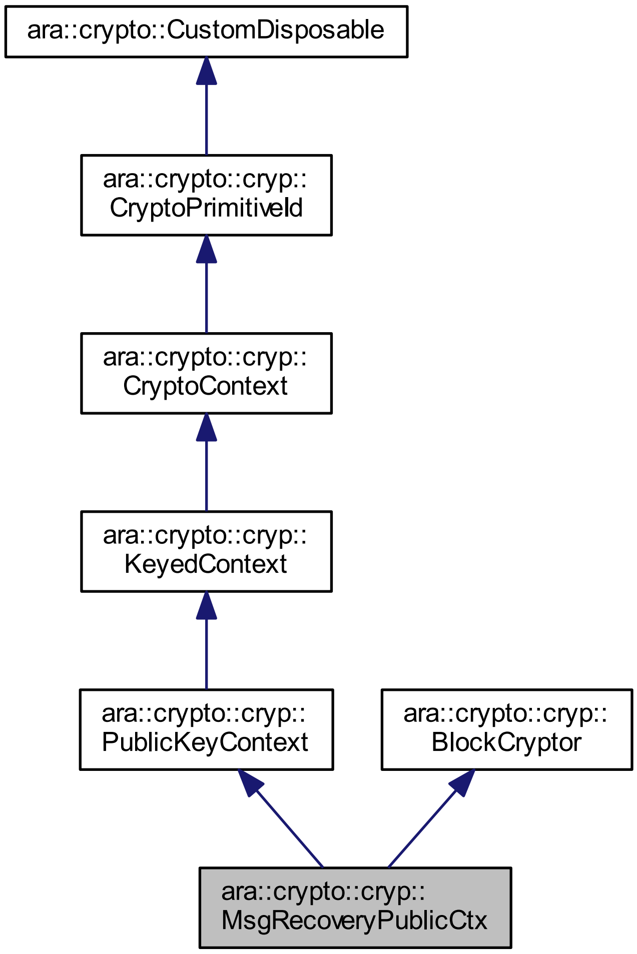
8.20.1.6 class ara::crypto::cryp::MsgRecoveryPublicCtx

[SWS_CRYPT_22200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::MsgRecoveryPublicCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::PublicKeyContext
Syntax:	`class MsgRecoveryPublicCtx : public PublicKeyContext {...};`
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).
Notes:	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)*

Inheritance diagram for ara::crypto::cryp::MsgRecoveryPublicCtx:



Public Types

- using Uptr = std::unique_ptr< MsgRecoveryPublicCtx, CustomDeleter >

Additional Inherited Members

8.20.1.6.1 Member Typedef Documentation

8.20.1.6.1.1 using ara::crypto::cryp::MsgRecoveryPublicCtx::Uptr = std::unique_ptr<MsgRecoveryPublicCtx, CustomDeleter>

[SWS_CRYPT_22201]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::MsgRecoveryPublicCtx::Uptr
Scope:	class ara::crypto::cryp::MsgRecoveryPublicCtx
Derived from:	std::unique_ptr<MsgRecoveryPublicCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::MsgRecoveryPublicCtx::Uptr = std::unique_ptr<MsgRecoveryPublicCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/msg_recovery_public_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

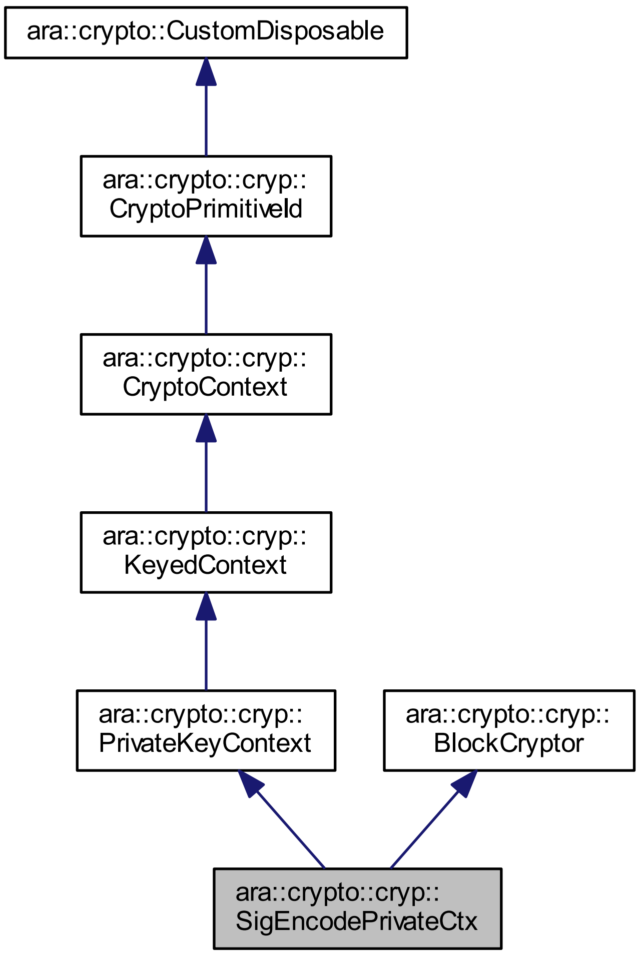
8.20.1.7 class ara::crypto::cryp::SigEncodePrivateCtx

[SWS_CRYPT_23200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::SigEncodePrivateCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::PrivateKeyContext
Syntax:	`class SigEncodePrivateCtx : public PrivateKeyContext {...};`
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).
Notes:	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)*

Inheritance diagram for ara::crypto::cryp::SigEncodePrivateCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::KeyedContext
ara::crypto::cryp::PrivateKeyContext
ara::crypto::cryp::BlockCryptor
ara::crypto::cryp::SigEncodePrivateCtx

Public Types

- using Uptr = std::unique_ptr< SigEncodePrivateCtx, CustomDeleter >

Additional Inherited Members

8.20.1.7.1 Member Typedef Documentation

8.20.1.7.1.1 using ara::crypto::cryp::SigEncodePrivateCtx::Uptr = std::unique_ptr<SigEncodePrivateCtx, CustomDeleter>

[SWS_CRYPT_23201]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::SigEncodePrivateCtx::Uptr
Scope:	class ara::crypto::cryp::SigEncodePrivateCtx
Derived from:	std::unique_ptr<SigEncodePrivateCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::SigEncodePrivateCtx::Uptr = std::unique_ptr<SigEncodePrivateCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/sig_encode_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

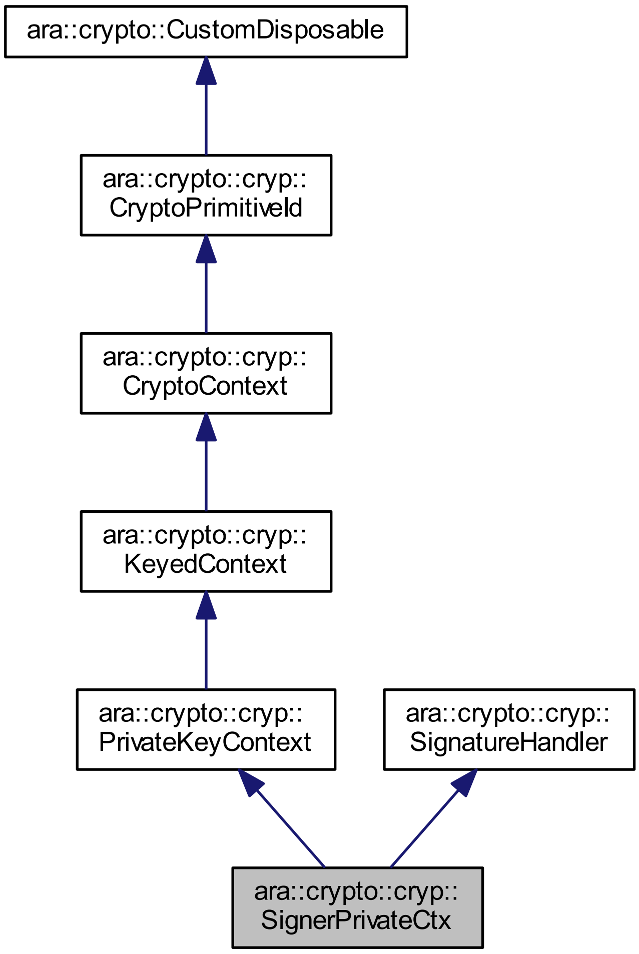
8.20.1.8 class ara::crypto::cryp::SignerPrivateCtx

[SWS_CRYPT_23500]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02204)*

Inheritance diagram for ara::crypto::cryp::SignerPrivateCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: PrivateKeyContext
ara::crypto::cryp:: SignatureHandler
ara::crypto::cryp:: SignerPrivateCtx

Public Types

- using Uptr = std::unique_ptr< SignerPrivateCtx, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< Signature::Uptrc > Sign (const HashFunctionCtx &hash, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0

- virtual ara::core::Result< std::size_t > Sign (WritableMemRegion signature, ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0
- template<typename Alloc = DefBytesAllocator>
  ara::core::Result< void > Sign (ByteVectorT< Alloc > &signature, ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept

Additional Inherited Members

8.20.1.8.1 Member Typedef Documentation

8.20.1.8.1.1 using ara::crypto::cryp::SignerPrivateCtx::Uptr = std::unique_ptr<SignerPrivateCtx, CustomDeleter>

[SWS_CRYPT_23501]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::SignerPrivateCtx::Uptr
Scope:	class ara::crypto::cryp::SignerPrivateCtx
Derived from:	std::unique_ptr<SignerPrivateCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::SignerPrivateCtx::Uptr = std::unique_ptr<SignerPrivateCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.20.1.8.2 Member Function Documentation

8.20.1.8.2.1 virtual ara::core::Result<Signature::Uptrc> ara::crypto::cryp::SignerPrivateCtx::Sign (const HashFunctionCtx & *hash*, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap, ReadOnlyMemRegion *context* = ReadOnlyMemRegion *()*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23511]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SignerPrivateCtx::Sign(const HashFunctionCtx &hash, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, ReadOnlyMemRegion context=ReadOnlyMemRegion())	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual ara::core::Result<Signature::Uptrc> Sign (const HashFunctionCtx &hash, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	hash	a finalized hash-function context that contains a digest value ready for sign
	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
	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	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Sign a provided digest value stored in the hash-function context.	
Notes:	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. [Error]: SecurityErrorDomain::kInvalidArgument if hash-function algorithm does not comply with the signature algorithm specification of this context [Error]: SecurityErrorDomain::kInvalidInputSize if the user supplied context has incorrect (or unsupported) size [Error]: SecurityErrorDomain::kProcessingNotFinished if the method hash.Finish() was not called before the call of this method [Error]: SecurityErrorDomain::kUninitializedContext this context was not initialized by a key value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02404)*

8.20.1.8.2.2 virtual ara::core::Result<std::size_t> ara::crypto::cryp::SignerPrivateCtx::Sign (WritableMemRegion *signature*, ReadOnlyMemRegion *value*, ReadOnlyMemRegion *context* = ReadOnlyMemRegion *()*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_23512]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::SignerPrivateCtx::Sign(WritableMemRegion signature, ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::SignerPrivateCtx	
Syntax:	`virtual ara::core::Result<std::size_t> Sign (WritableMemRegion signature, ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	value	the (pre-)hashed or direct message value that should be signed
	context	an optional user supplied "context" (its support depends from concrete algorithm)
Parameters (out):	signature	a buffer for the resulting signature
Return value:	ara::core::Result< std::size_t >	actual size of the signature value stored to the output buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Sign a directly provided hash or message value.	
Notes:	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). [Error]: SecurityErrorDomain::kInvalidInputSize if size of the input value or context arguments are incorrect / unsupported [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the output signature buffer is not enough [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value	

⌋*(RS_CRYPTO_02404)*

8.20.1.8.2.3 template<typename Alloc = DefBytesAllocator> ara::core::Result<void> ara::crypto::cryp::SignerPrivateCtx::Sign (ByteVectorT< Alloc > & *signature*, ReadOnlyMemRegion *value*, ReadOnlyMemRegion *context* = ReadOnlyMemRegion *()*) const `[noexcept]`

[SWS_CRYPT_23513]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::SignerPrivateCtx::Sign(ByteVectorT< Alloc > &signature, ReadOnlyMemRegion value, ReadOnlyMemRegion context=ReadOnlyMemRegion()
Scope:	class ara::crypto::cryp::SignerPrivateCtx

Syntax:	`template <typename Alloc>`
	`inline ara::core::Result<void> Sign (ByteVectorT< Alloc > &signature, 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)
Parameters (out):	signature	pre-reserved managed container for resulting signature
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/signer_private_ctx.h"	
Description:	Sign a directly provided hash or message value.	
Notes:	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! [Error]: SecurityErrorDomain::kInvalidInputSize if size of the input value or context arguments are incorrect / unsupported [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the output signature container is not enough [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value	

⌋*(RS_CRYPTO_02404)*

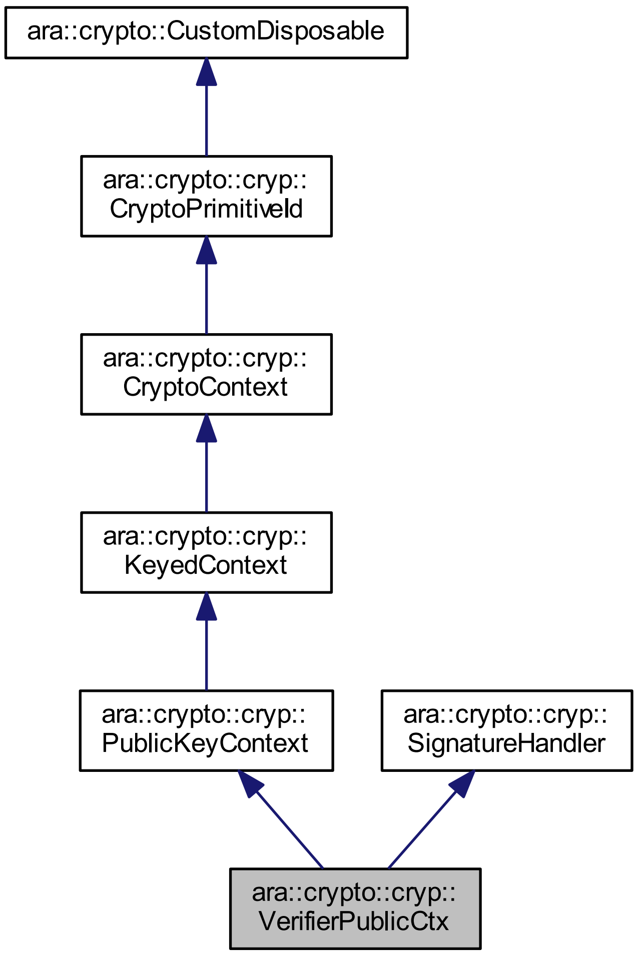
8.20.1.9 class ara::crypto::cryp::VerifierPublicCtx

[SWS_CRYPT_24100]{DRAFT} ⌈

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

⌋*(RS_CRYPTO_02204)*

Inheritance diagram for ara::crypto::cryp::VerifierPublicCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp::CryptoPrimitiveId
ara::crypto::cryp::CryptoContext
ara::crypto::cryp::KeyedContext
ara::crypto::cryp::PublicKeyContext
ara::crypto::cryp::SignatureHandler
ara::crypto::cryp::VerifierPublicCtx

Public Types

- using Uptr = std::unique_ptr< VerifierPublicCtx, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< bool > Verify (const HashFunctionCtx &hash, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0

- virtual ara::core::Result< bool > Verify (ReadOnlyMemRegion value, ReadOnlyMemRegion signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0

Additional Inherited Members

8.20.1.9.1 Member Typedef Documentation

8.20.1.9.1.1 using ara::crypto::cryp::VerifierPublicCtx::Uptr = std::unique_ptr<VerifierPublicCtx, CustomDeleter>

[SWS_CRYPT_24101]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::VerifierPublicCtx::Uptr
Scope:	class ara::crypto::cryp::VerifierPublicCtx
Derived from:	std::unique_ptr<VerifierPublicCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::VerifierPublicCtx::Uptr = std::unique_ptr<VerifierPublicCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.20.1.9.2 Member Function Documentation

8.20.1.9.2.1 virtual ara::core::Result<bool> ara::crypto::cryp::VerifierPublicCtx::Verify (const HashFunctionCtx & *hash*, const Signature & *signature*, ReadOnlyMemRegion *context* = ReadOnlyMemRegion*()*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24111]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::VerifierPublicCtx::Verify(const HashFunctionCtx &hash, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::VerifierPublicCtx	
Syntax:	`virtual ara::core::Result<bool> Verify (const HashFunctionCtx &hash, const Signature &signature, ReadOnlyMemRegion context=ReadOnlyMemRegion()) const noexcept=0;`	
Parameters (in):	hash	the finalized hash-function context that contains a digest value ready for the verification

▽

△

	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	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Verify signature by a digest value stored in the hash-function context.	
Notes:	The user supplied context may be used for such algorithms as: Ed25519ctx, Ed25519ph, Ed448ph. If any of the 5 mentioned below conditions was violated then this method returns false: This method must control compliance of the real hash-function algorithm to the hash algorithm specified in the signature!This method must check equality of the real public key COUID and corresponded dependency COUID in the signature!This method must control the hash-function algorithm ID compliance to the signature algorithm specification of the context!This method must control compliance of the signature algorithm ID in the signature object and the signature verification context!This method must control compliance of the signature algorithm ID in the signature object and the user supplied context size! [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kIncompatibleObject if algorithms of hash or signature arguments are incompatible with the context configuration [Error]: SecurityErrorDomain::kIncompatibleArguments if algorithms of hash or signature arguments are compatible with the context configuration, but incompatible between each other (it can be a case if the verifier context configuration doesn't restrict the hash function algorithm) [Error]: SecurityErrorDomain::kBadObjectReference if provided signature object references to an instance of a public key different from the one loaded to the context, i.e. if the COUID of the public key in the context (see this->GetActualKeyBitLength()) is not equal to the COUID referenced from the signature object (see signature.HasDependence()) [Error]: SecurityErrorDomain::kProcessingNotFinished if the method hash.Finish() was not called before this method call [Error]: SecurityErrorDomain::kInvalidInputSize if the context argument has unsupported size	

⌋*(RS_CRYPTO_02311)*

8.20.1.9.2.2 virtual ara::core::Result<bool> ara::crypto::cryp::VerifierPublicCtx::Verify (ReadOnlyMemRegion *value*, ReadOnlyMemRegion *signature*, ReadOnlyMemRegion *context* = ReadOnlyMemRegion*()*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24112]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::cryp::VerifierPublicCtx::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
	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	
Header file:	#include "ara/crypto/cryp/verifier_public_ctx.h"	
Description:	Verify signature by a directly provided hash or message value.	
Notes:	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 a size of the value, signature or context BLOB is incorrect then this method returns false before starting of any calculations! [Error]: SecurityErrorDomain::kUninitializedContext if the context was not initialized by a key value [Error]: SecurityErrorDomain::kInvalidInputSize if the value, signature or context has unsupported size	

⌋*(RS_CRYPTO_02311)*

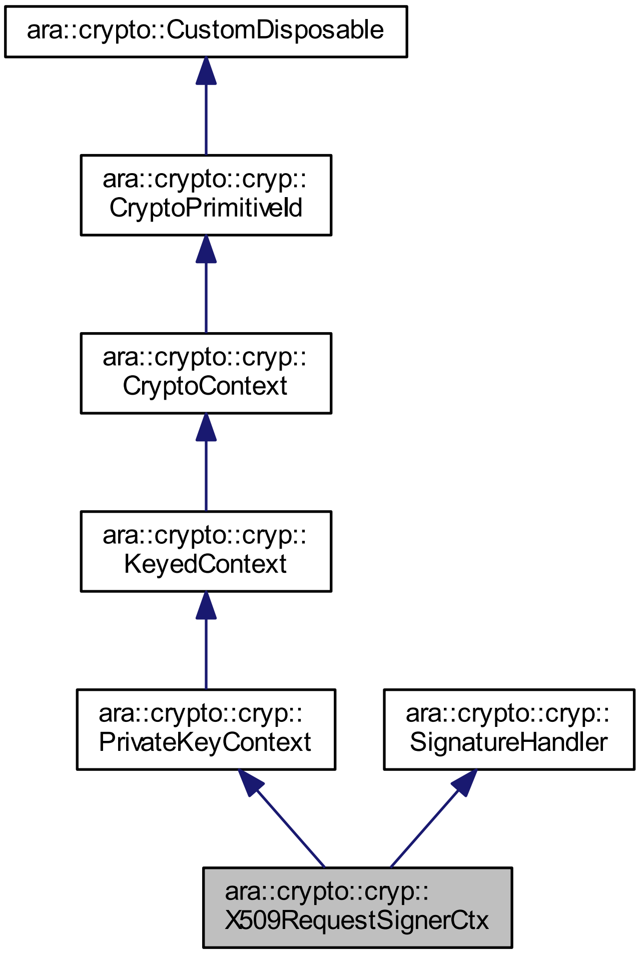
8.20.1.10 class ara::crypto::cryp::X509RequestSignerCtx

[SWS_CRYPT_24500]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::cryp::X509RequestSignerCtx
Scope:	namespace ara::crypto::cryp
Base class:	ara::crypto::cryp::PrivateKeyContext
Syntax:	`class X509RequestSignerCtx : public PrivateKeyContext {...};`
Header file:	#include "ara/crypto/cryp/x509_request_signer_ctx.h"
Description:	X.509 Request Signer Context interface.
Notes:	Any private key (including keys without Key::kAllowSignature attribute) can be loaded to this interface context!

⌋*(RS_CRYPTO_02307)*

Inheritance diagram for ara::crypto::cryp::X509RequestSignerCtx:


ara::crypto::CustomDisposable
ara::crypto::cryp:: CryptoPrimitiveId
ara::crypto::cryp:: CryptoContext
ara::crypto::cryp:: KeyedContext
ara::crypto::cryp:: PrivateKeyContext
ara::crypto::cryp:: SignatureHandler
ara::crypto::cryp:: X509RequestSignerCtx

Public Types

- using Uptr = std::unique_ptr< X509RequestSignerCtx, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< X509CertRequest::Uptrc > CreateCertRequest (ReadOnlyMemRegion derSubjectDN, ReadOnlyMemRegion x509Extensions=ReadOnlyMemRegion(), unsigned version=1, ReservedObjectIndex reservedIndex=kAllocObjectOnHeap) const noexcept=0

Additional Inherited Members

8.20.1.10.1 Member Typedef Documentation

8.20.1.10.1.1 using ara::crypto::cryp::X509RequestSignerCtx::Uptr = std::unique_ptr<X509RequestSignerCtx, CustomDeleter>

[SWS_CRYPT_24501]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::cryp::X509RequestSignerCtx::Uptr
Scope:	class ara::crypto::cryp::X509RequestSignerCtx
Derived from:	std::unique_ptr<X509RequestSignerCtx, CustomDeleter>
Syntax:	`using ara::crypto::cryp::X509RequestSignerCtx::Uptr = std::unique_ptr<X509RequestSignerCtx, CustomDeleter>;`
Header file:	#include "ara/crypto/cryp/x509_request_signer_ctx.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.20.1.10.2 Member Function Documentation

8.20.1.10.2.1 virtual ara::core::Result<X509CertRequest::Uptrc> ara::crypto::cryp::X509RequestSignerCtx::CreateCertRequest (ReadOnlyMemRegion *derSubjectDN*, ReadOnlyMemRegion *x509Extensions* = ReadOnlyMemRegion`()`, unsigned *version* = `1`, ReservedObjectIndex *reservedIndex* = kAllocObjectOnHeap) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_24511]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::cryp::X509RequestSignerCtx::CreateCertRequest(ReadOnlyMemRegion der SubjectDN, ReadOnlyMemRegion x509Extensions=ReadOnlyMemRegion()	
Scope:	class ara::crypto::cryp::X509RequestSignerCtx	
Syntax:	`virtual ara::core::Result<X509CertRequest::Uptrc> CreateCertRequest (ReadOnlyMemRegion derSubjectDN, ReadOnlyMemRegion x509Extensions=Read OnlyMemRegion(), unsigned version=1, ReservedObjectIndex reserved Index=kAllocObjectOnHeap) const noexcept=0;`	
Parameters (in):	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

▽

△

	reservedIndex	an optional index of reserved Object slot that should be used for this allocation or default marker, which says to allocate on the heap
Return value:	ara::core::Result< X509Cert Request::Uptrc >	unique smart pointer to created certification request
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/cryp/x509_request_signer_ctx.h"	
Description:	Create certification request for a private key loaded to the context.	
Notes:	[Error]: SecurityErrorDomain::kUnexpectedValue if any of arguments has incorrect/unsupported value [Error]: SecurityErrorDomain::kBusyResource if the slot specified by reservedIndex is busy yet [Error]: SecurityErrorDomain::kUnreservedResource if the slot specified by reservedIndex was not allocated [Error]: SecurityErrorDomain::kInsufficientResource if capacity of slot specified by reserved Index is not enough for placing of the target object [Error]: SecurityErrorDomain::kBadAlloc if (reservedIndex == kAllocObjectOnHeap), but allocation memory on the heap is impossible	

⌋*(RS_CRYPTO_02306, RS_CRYPTO_02404)*

8.21 Key Storage Provider API

Namespaces

- ara::crypto::keys

Classes

- struct ara::crypto::keys::KeySlotContentProps
- struct ara::crypto::keys::KeySlotPrototypeProps
- class ara::crypto::keys::KeyStorageProvider
- class ara::crypto::keys::UpdatesObserver
- struct ara::crypto::keys::UserPermissions

Typedefs

- using ara::crypto::keys::SlotNumber = std::size_t
- using ara::crypto::keys::TransactionId = std::uint64_t
- using ara::crypto::keys::TransactionScope = ara::core::Vector< SlotNumber >

Enumerations

Functions

- ara::core::Result< KeyStorageProvider::Sptr > ara::crypto::keys::LoadKeyStorageProvider () noexcept
- constexpr bool ara::crypto::keys::operator== (const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) noexcept
- constexpr bool ara::crypto::keys::operator!= (const KeySlotContentProps &lhs, const KeySlotContentProps &rhs) noexcept
- constexpr bool ara::crypto::keys::operator== (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept
- constexpr bool ara::crypto::keys::operator!= (const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs) noexcept
- virtual void ara::crypto::keys::UpdatesObserver::OnUpdate (const TransactionScope &updatedSlots) noexcept=0
- constexpr bool ara::crypto::keys::operator== (const UserPermissions &lhs, const UserPermissions &rhs) noexcept

- constexpr bool ara::crypto::keys::operator!= (const UserPermissions &lhs, const UserPermissions &rhs) noexcept

Variables

- const SlotNumber ara::crypto::keys::kInvalidSlot = static_cast<SlotNumber>(-1LL)

Detailed Description

Key Storage Provider represents logically single entry to whole functionality of secure persistent storage and access control for all cryptographic objects. Crypto Stack provides a unified Key Storage Provider for access to all secure storage space independently from actual physical locations of it's specific partitions (on file system or inside some HSMs). All public interfaces of the Key Storage Provider must be defined in this namespace.

8.21.1 Class Documentation

8.21.1.1 struct ara::crypto::keys::KeySlotContentProps

[SWS_CRYPT_30500]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::crypto::keys::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.
Notes:	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, RS_CRYPTO_02404)*

Public Attributes

- CryptoObjectUid mObjectUid
- CryptoObjectUid mDependencyUid
- CryptoAlgId mAlgId
- SlotNumber mDependencySlot
- std::size_t mObjectSize

- AllowedUsageFlags mAllowedUsage
- std::uint32_t mReferencesCounter
- CryptoObjectType mObjectType
- bool mExportability

8.21.1.1.1 Member Data Documentation

8.21.1.1.1.1 CryptoObjectUid ara::crypto::keys::KeySlotContentProps::mObjectUid

[SWS_CRYPT_30501]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mObjectUid
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	CryptoObjectUid
Syntax:	`CryptoObjectUid ara::crypto::keys::KeySlotContentProps::mObjectUid;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	UID of a Crypto Object stored to the slot.

⌋*(RS_CRYPTO_02111)*

8.21.1.1.1.2 CryptoObjectUid ara::crypto::keys::KeySlotContentProps::mDependencyUid

[SWS_CRYPT_30502]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mDependencyUid
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	CryptoObjectUid
Syntax:	`CryptoObjectUid ara::crypto::keys::KeySlotContentProps::mDependency Uid;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	UID of another Crypto Object on which the object stored in this slot depends.

⌋*(RS_CRYPTO_02111)*

8.21.1.1.1.3 CryptoAlgId ara::crypto::keys::KeySlotContentProps::mAlgId

[SWS_CRYPT_30503]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mAlgId
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	CryptoAlgId
Syntax:	`CryptoAlgId ara::crypto::keys::KeySlotContentProps::mAlgId;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Cryptoalgorithm of actual object stored to the slot.

⌋*(RS_CRYPTO_02111)*

8.21.1.1.1.4 SlotNumber ara::crypto::keys::KeySlotContentProps::mDependencySlot

[SWS_CRYPT_30504]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mDependencySlot
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	SlotNumber
Syntax:	`SlotNumber ara::crypto::keys::KeySlotContentProps::mDependencySlot;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Optional number of a slot where the dependency object is stored, if (mDependencyUid.IsNil() == false)

⌋*(RS_CRYPTO_02111)*

8.21.1.1.1.5 std::size_t ara::crypto::keys::KeySlotContentProps::mObjectSize

[SWS_CRYPT_30505]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mObjectSize
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	std::size_t
Syntax:	`std::size_t ara::crypto::keys::KeySlotContentProps::mObjectSize;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Actual size of an object currently stored to the slot.

⌋*(RS_CRYPTO_02111)*

8.21.1.1.1.6 AllowedUsageFlags ara::crypto::keys::KeySlotContentProps::mAllowedUsage

[SWS_CRYPT_30506]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mAllowedUsage
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	AllowedUsageFlags
Syntax:	`AllowedUsageFlags ara::crypto::keys::KeySlotContentProps::mAllowedUsage;`
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)*

8.21.1.1.1.7 std::uint32_t ara::crypto::keys::KeySlotContentProps::mReferencesCounter

[SWS_CRYPT_30507]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mReferencesCounter
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	std::uint32_t
Syntax:	`std::uint32_t ara::crypto::keys::KeySlotContentProps::mReferencesCounter;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Number of references to this object instance from other slots in the Key Storage.

⌋*(RS_CRYPTO_02111)*

8.21.1.1.1.8 CryptoObjectType ara::crypto::keys::KeySlotContentProps::mObjectType

[SWS_CRYPT_30508]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mObjectType
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	CryptoObjectType
Syntax:	`CryptoObjectType ara::crypto::keys::KeySlotContentProps::mObjectType;`

▽

△

Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Actual type of an object stored to the slot.

⌋*(RS_CRYPTO_02111)*

8.21.1.1.1.9 bool ara::crypto::keys::KeySlotContentProps::mExportability

[SWS_CRYPT_30509]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotContentProps::mExportability
Scope:	struct ara::crypto::keys::KeySlotContentProps
Type:	bool
Syntax:	`bool ara::crypto::keys::KeySlotContentProps::mExportability;`
Header file:	#include "ara/crypto/keys/key_slot_content_props.h"
Description:	Effective value of the exportability attribute of current object.

⌋*(RS_CRYPTO_02111)*

8.21.1.2 struct ara::crypto::keys::KeySlotPrototypeProps

[SWS_CRYPT_30300]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::crypto::keys::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, RS_CRYPTO_02404)*

Public Attributes

- LogicalSlotUid mLogicalSlotUid
- LogicalSlotUid mDependencySlotUid
- ActorUid mOwnerUid
- CryptoProviderUid mCryptoProviderUid
- CryptoObjectUid mVersionTrack

- CryptoAlgId mAlgId
- std::size_t mSlotCapacity
- CryptoObjectType mObjectType
- CryptoObjectType mDependecyType
- VersionControlType mVersionControl
- bool mExportability

8.21.1.2.1 Member Data Documentation

8.21.1.2.1.1 LogicalSlotUid ara::crypto::keys::KeySlotPrototypeProps::mLogicalSlotUid

[SWS_CRYPT_30301]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mLogicalSlotUid
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	LogicalSlotUid
Syntax:	`LogicalSlotUid ara::crypto::keys::KeySlotPrototypeProps::mLogicalSlotUid;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Logical Slot UID defined at Design phase of the Owner software.

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.2 LogicalSlotUid ara::crypto::keys::KeySlotPrototypeProps::mDependencySlotUid

[SWS_CRYPT_30302]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mDependencySlotUid
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	LogicalSlotUid
Syntax:	`LogicalSlotUid ara::crypto::keys::KeySlotPrototypeProps::mDependencySlotUid;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Optional reference to Dependency Logical Slot UID defined at Design phase of the Owner software.

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.3 ActorUid ara::crypto::keys::KeySlotPrototypeProps::mOwnerUid

[SWS_CRYPT_30303]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mOwnerUid
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	ActorUid
Syntax:	`ActorUid ara::crypto::keys::KeySlotPrototypeProps::mOwnerUid;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	UID of an "Owner Actor" of this key slot.

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.4 CryptoProviderUid ara::crypto::keys::KeySlotPrototypeProps::mCryptoProviderUid

[SWS_CRYPT_30304]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mCryptoProviderUid
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	CryptoProviderUid
Syntax:	`CryptoProviderUid ara::crypto::keys::KeySlotPrototypeProps::mCrypto ProviderUid;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	UUID of a Crypto Provider assigned for this key slot processing (at Integration stage).

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.5 CryptoObjectUid ara::crypto::keys::KeySlotPrototypeProps::mVersionTrack

[SWS_CRYPT_30305]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mVersionTrack
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	CryptoObjectUid
Syntax:	`CryptoObjectUid ara::crypto::keys::KeySlotPrototypeProps::mVersion Track;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"

▽

△

Description:	COUID of last object stored to this slot (versionTrack.generatorUid restricts objects' source).

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.6 CryptoAlgId ara::crypto::keys::KeySlotPrototypeProps::mAlgId

[SWS_CRYPT_30306]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mAlgId
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	CryptoAlgId
Syntax:	`CryptoAlgId ara::crypto::keys::KeySlotPrototypeProps::mAlgId;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Cryptoalgorithm restriction (kAlgIdAny means without restriction). The algorithm can be specified partially: family & length, mode, padding.

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.7 std::size_t ara::crypto::keys::KeySlotPrototypeProps::mSlotCapacity

[SWS_CRYPT_30307]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mSlotCapacity
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	std::size_t
Syntax:	`std::size_t ara::crypto::keys::KeySlotPrototypeProps::mSlotCapacity;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Capacity of the slot in bytes.

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.8 CryptoObjectType ara::crypto::keys::KeySlotPrototypeProps::mObjectType

[SWS_CRYPT_30308]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mObjectType
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	CryptoObjectType
Syntax:	`CryptoObjectType ara::crypto::keys::KeySlotPrototypeProps::mObject Type;`
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)*

8.21.1.2.1.9 CryptoObjectType ara::crypto::keys::KeySlotPrototypeProps::mDependecyType

[SWS_CRYPT_30309]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mDependecyType
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	CryptoObjectType
Syntax:	`CryptoObjectType ara::crypto::keys::KeySlotPrototypeProps::mDependecy Type;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Type of object from which depends type prototyped for this slot. If this field contains Crypto ObjectType::kNone then this slot doesn't have any dependency.

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.10 VersionControlType ara::crypto::keys::KeySlotPrototypeProps::mVersionControl

[SWS_CRYPT_30310]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mVersionControl
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	VersionControlType
Syntax:	`VersionControlType ara::crypto::keys::KeySlotPrototypeProps::mVersion Control;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Version Control Type selects a rule restriction source of stored objects.

⌋*(RS_CRYPTO_02110)*

8.21.1.2.1.11 bool ara::crypto::keys::KeySlotPrototypeProps::mExportability

[SWS_CRYPT_30311]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::KeySlotPrototypeProps::mExportability
Scope:	struct ara::crypto::keys::KeySlotPrototypeProps
Type:	bool
Syntax:	`bool ara::crypto::keys::KeySlotPrototypeProps::mExportability;`
Header file:	#include "ara/crypto/keys/key_slot_prototype_props.h"
Description:	Exportability restriction: if false then any exportable object cannot be saved to this slot.

⌋*(RS_CRYPTO_02110)*

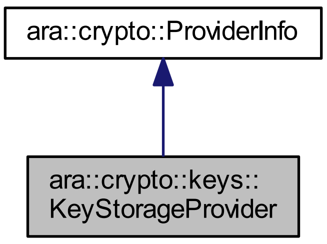
8.21.1.3 class ara::crypto::keys::KeyStorageProvider

[SWS_CRYPT_30100]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::keys::KeyStorageProvider
Scope:	namespace ara::crypto::keys
Base class:	std::enable_shared_from_this< KeyStorageProvider >
Syntax:	`class KeyStorageProvider : public enable_shared_from_this< KeyStorage Provider > {...};`
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	Key Storage Provider interface.
Notes:	Any object is uniquely identified by the combination of its GUID 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)*

Inheritance diagram for ara::crypto::keys::KeyStorageProvider:


ara::crypto::ProviderInfo
ara::crypto::keys::KeyStorageProvider

Public Types

- using Sptr = std::shared_ptr< KeyStorageProvider >
- using ObjectUid = CryptoObjectUid
- using SlotUid = LogicalSlotUid
- using ContentType = CryptoObjectType

Public Member Functions

- virtual ~KeyStorageProvider () noexcept=default
- virtual SlotNumber FindSlot (const SlotUid &slotUid, CryptoProviderUid *providerUid=nullptr) const noexcept=0
- virtual SlotNumber FindObject (const ObjectUid &objectUid, ContentType objectType, CryptoProviderUid &providerUid, SlotNumber previousFound=kInvalidSlot) const noexcept=0
- virtual ara::core::Result< bool > IsEmpty (SlotNumber slotNum) const noexcept=0
- virtual ara::core::Result< TrustedContainer::Uptrc > OpenAsUser (SlotNumber slotNum, bool subscribeForUpdates=false) noexcept=0
- virtual ara::core::Result< TrustedContainer::Uptr > OpenAsOwner (SlotNumber slotNum) noexcept=0
- virtual ara::core::Result< void > SaveCopy (SlotNumber slotNum, const TrustedContainer &container) noexcept=0
- virtual ara::core::Result< void > Clear (SlotNumber slotNum) noexcept=0
- virtual ara::core::Result< void > GetPrototypedProps (SlotNumber slotNum, KeySlotPrototypeProps &props) const noexcept=0
- virtual ara::core::Result< void > GetContentProps (SlotNumber slotNum, KeySlotContentProps &props) const noexcept=0
- virtual ara::core::Result< void > GetDefaultCryptoProviderUid (SlotNumber slotNum, CryptoProviderUid &providerUid) const noexcept=0
- virtual ara::core::Result< void > GetOwner (SlotNumber slotNum, ActorUid &ownerUid) const noexcept=0
- virtual ara::core::Result< std::size_t > GetUsers (SlotNumber slotNum, ara::core::Vector< UserPermissions > *users=nullptr) const noexcept=0
- virtual ara::core::Result< TransactionId > BeginTransaction (const TransactionScope &targetSlots) noexcept=0
- virtual ara::core::Result< void > CommitTransaction (TransactionId id) noexcept=0

- virtual ara::core::Result< void > RollbackTransaction (TransactionId id) noexcept=0
- virtual ara::core::Result< void > UnsubscribeObserver (SlotNumber slot) noexcept=0
- virtual SlotNumber FindReferringSlot (SlotNumber targetSlot, SlotNumber previousFound=kInvalidSlot) const noexcept=0
- virtual ara::core::Result< void > ResetReference (SlotNumber referrerSlot, SlotNumber referencedSlot=kInvalidSlot) const noexcept=0
- virtual ara::core::Result< bool > CanLoadToCryptoProvider (SlotNumber slotNum, const CryptoProviderUid &providerUid) const noexcept=0
- virtual UpdatesObserver::Sptr RegisterObserver (UpdatesObserver::Sptr observer=nullptr) noexcept=0
- virtual UpdatesObserver::Sptr GetRegisteredObserver () const noexcept=0
- virtual ara::core::Result< SlotNumber > FindSlot (const ara::core::InstanceSpecifier &slotSpecifier, CryptoProviderUid &providerUid) const noexcept=0

Additional Inherited Members

8.21.1.3.1 Member Typedef Documentation

8.21.1.3.1.1 using ara::crypto::keys::KeyStorageProvider::Sptr = std::shared_ptr<KeyStorageProvider>

[SWS_CRYPT_30101]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::KeyStorageProvider::Sptr
Scope:	class ara::crypto::keys::KeyStorageProvider
Derived from:	std::shared_ptr<KeyStorageProvider>
Syntax:	`using ara::crypto::keys::KeyStorageProvider::Sptr = std::shared_ptr<KeyStorageProvider>;`
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.21.1.3.1.2 using ara::crypto::keys::KeyStorageProvider::ObjectUid = CryptoObjectUid

[SWS_CRYPT_30102]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::KeyStorageProvider::ObjectUid
Scope:	class ara::crypto::keys::KeyStorageProvider
Derived from:	CryptoObjectUid
Syntax:	`using ara::crypto::keys::KeyStorageProvider::ObjectUid = CryptoObjectUid;`
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	Definition of an object UID type.

⌋*(RS_CRYPTO_02005)*

8.21.1.3.1.3 using ara::crypto::keys::KeyStorageProvider::SlotUid = LogicalSlotUid

[SWS_CRYPT_30103]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::KeyStorageProvider::SlotUid
Scope:	class ara::crypto::keys::KeyStorageProvider
Derived from:	LogicalSlotUid
Syntax:	`using ara::crypto::keys::KeyStorageProvider::SlotUid = LogicalSlotUid;`
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	Definition of an object UID type.

⌋*(RS_CRYPTO_02404, RS_CRYPTO_02405)*

8.21.1.3.1.4 using ara::crypto::keys::KeyStorageProvider::ContentType = CryptoObjectType

[SWS_CRYPT_30104]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::KeyStorageProvider::ContentType
Scope:	class ara::crypto::keys::KeyStorageProvider
Derived from:	CryptoObjectType
Syntax:	`using ara::crypto::keys::KeyStorageProvider::ContentType = CryptoObjectType;`
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	Definition of a slot content type.

⌋*(RS_CRYPTO_02311)*

8.21.1.3.2 Constructor & Destructor Documentation

8.21.1.3.2.1 virtual ara::crypto::keys::KeyStorageProvider::~KeyStorageProvider ( ) [virtual], [default], [noexcept]

[SWS_CRYPT_30110]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::keys::KeyStorageProvider::~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_02311)*

8.21.1.3.3 Member Function Documentation

8.21.1.3.3.1 virtual SlotNumber ara::crypto::keys::KeyStorageProvider::FindSlot (const SlotUid & *slotUid,* CryptoProviderUid ∗ *providerUid* = *nullptr*) const [pure virtual], [noexcept]

[SWS_CRYPT_30111]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::FindSlot(const SlotUid &slotUid, CryptoProviderUid *providerUid=nullptr)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual SlotNumber FindSlot (const SlotUid &slotUid, CryptoProviderUid *providerUid=nullptr) const noexcept=0;`	
Parameters (in):	slotUid	Logic Slot UID for search
Parameters (out):	providerUid	an optional pointer to UID of default Crypto Provider assigned for servicing this slot
Return value:	SlotNumber	number of found slot or kInvalidSlot if a slot with such Logic UID was not found
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Find a slot number by the Logic (persistent) Slot UID.	
Notes:	If the returned providerUid has nil value then the slot content can be loaded to any Crypto Provider!	

⌋*(RS_CRYPTO_02405)*

8.21.1.3.3.2 virtual SlotNumber ara::crypto::keys::KeyStorageProvider::FindObject (const ObjectUid & *objectUid,* ContentType *objectType,* CryptoProviderUid & *providerUid,* SlotNumber *previousFound* = kInvalidSlot) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30112]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::FindObject(const ObjectUid &objectUid, ContentType objectType, CryptoProviderUid &providerUid, SlotNumber previousFound=kInvalidSlot)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual SlotNumber FindObject (const ObjectUid &objectUid, ContentType objectType, CryptoProviderUid &providerUid, SlotNumber previousFound=kInvalidSlot) const noexcept=0;`	
Parameters (in):	objectUid	target object UID
	objectType	type of the target object
	previousFound	the number of previous found key slot (the search will start from next slot number)
Parameters (inout):	providerUid	the UID of Crypto Provider responsible for servicing of the slot (a non-zero input value restricts a search scope to specific Crypto Provider)
Return value:	SlotNumber	number of a slot containing the found object or kInvalidSlot if the object was not found
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Find a slot number by the Crypto Object's UID and type.	
Notes:	Use previousFound = kInvalidSlot to start the search from the begin. If the provider UID has the zero value then the search is executed through whole Key Storage (without limitation to any specific Crypto Provider). If an application needs to find all instances of a crypto objects (through all Crypto Providers) then this method should be called multiple times until it will return kInvalidSlot. If the returned providerUid has nil value then the slot content can be loaded to any Crypto Provider! Also see cryp::CryptoObject::Save() for restrictions of content saving to a key slot.	

⌋*(RS_CRYPTO_02004, RS_CRYPTO_02005)*

8.21.1.3.3.3 virtual ara::core::Result<bool> ara::crypto::keys::KeyStorageProvider::IsEmpty (SlotNumber *slotNum*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30113]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::IsEmpty(SlotNumber slotNum)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<bool> IsEmpty (SlotNumber slotNum) const noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Return value:	ara::core::Result< bool >	true if the slot is empty or false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Check the slot for emptiness.	
Notes:	If the specified slot is involved to a proceeding transaction then the status of the "User" visible part should be returned! [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which has no any ("Owner" or "User") access rights to the key slot	

⌋*(RS_CRYPTO_02311)*

8.21.1.3.3.4 virtual ara::core::Result<TrustedContainer::Uptrc> ara::crypto::keys::KeyStorageProvider::OpenAsUser (SlotNumber *slotNum*, bool *subscribeForUpdates* = *false*) [pure virtual], [noexcept]

[SWS_CRYPT_30114]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::OpenAsUser(SlotNumber slotNum, bool subscribeForUpdates=false)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<TrustedContainer::Uptrc> OpenAsUser (SlotNumber slotNum, bool subscribeForUpdates=false) noexcept=0;`	
Parameters (in):	slotNum	target slot number
	subscribeForUpdates	if this flag is true then the UpdatesObserver instance supplied to the Key Storage factory function will be subscribed for updates of the slotNum key slot
Return value:	ara::core::Result< TrustedContainer::Uptrc >	an unique smart pointer to allocated trusted container associated with the slot content
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Open a slot containing an existing object with "User" permissions and associate a trusted container to it (suitable for reading only).	

▽

△

Notes:	Only a non-empty slot may be opened by this method! 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 Crypto Object 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). [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (i.e. the slot is not allocated) [Error]: SecurityErrorDomain::kEmptyContainer if the slot is empty [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which has no any "User" access rights to the key slot

⌋*(RS_CRYPTO_02311)*

8.21.1.3.3.5 virtual ara::core::Result<TrustedContainer::Uptr> ara::crypto::keys::KeyStorageProvider::OpenAsOwner (SlotNumber *slotNum*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30115]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::OpenAsOwner(SlotNumber slotNum)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<TrustedContainer::Uptr> OpenAsOwner (SlotNumber slotNum) noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Return value:	ara::core::Result< TrustedContainer::Uptr >	an unique smart pointer to allocated trusted container associated to the slot's space
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Open a key slot with "Owner" permissions and associate a trusted container with it for exclusive access.	
Notes:	Only single instance of the "Owner" TrustedContainer may exist for a key slot simultaneously! Slots opened by this method are not monitored by the UpdateObserver notification mechanism! [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kBusyResource if the specified slot is busy, i.e. another instance of the "Owner" TrustedContainer already exists [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which is not "Owner" of the key slot	

⌋*(RS_CRYPTO_02311)*

8.21.1.3.3.6 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::SaveCopy (SlotNumber *slotNum*, const TrustedContainer & *container*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30116]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::SaveCopy(SlotNumber slotNum, const TrustedContainer &container)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> SaveCopy (SlotNumber slotNum, const TrustedContainer &container) noexcept=0;`	
Parameters (in):	slotNum	the target slot number
	container	the source volatile container
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Save a content of provided source trusted container to a persistent slot by their "Owner".	
Notes:	The source container may represent as a temporary (volatile) container, so a persistent slot in the Key Storage, but in the second case the calling application must be "Owner" of the source container too! This method may be used for atomic update of a key slot scoped to some transaction. In such case the "User" visible part of the slot will be updated only after correspondent call of CommitTransaction(). [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which is not "Owner" of the target key slot or source container (if it is a persistent container) [Error]: SecurityErrorDomain::kBusyResource if the target slot is opened by its "Owner", i.e. a trusted container returned by a call OpenAsOwner(slotNum) is not destroyed yet [Error]: SecurityErrorDomain::kIncompatibleObject if an object in the container is "session" [Error]: SecurityErrorDomain::kEmptyContainer if the source trusted container is empty [Error]: SecurityErrorDomain::kContentRestrictions if an object in the container doesn't satisfy the slot restrictions (including version control) [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated)	

⌋*(RS_CRYPTO_02311)*

8.21.1.3.3.7 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::Clear (SlotNumber *slotNum*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30117]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::Clear(SlotNumber slotNum)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> Clear (SlotNumber slotNum) noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Clear the slot identified by its number.	
Notes:	This method must perform a secure cleanup without the ability to restore the object data! If an object stored in the container references to another key slot, then "references counter" of the referenced key slot must be automatically decremented (after successful deleting of this object). This method may be used for atomic update of a key slot scoped to some transaction. In such case the "User" visible part of the slot will be updated only after correspondent call of Commit Transaction(). [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kBusyResource if the target slot is opened by its "Owner", i.e. a trusted container returned by a call OpenAsOwner(slotNum) is not destroyed yet [Error]: SecurityErrorDomain::kLockedByReference if the internal "references counter" of this slot has non-zero value [Error]: SecurityErrorDomain::kBadObjectReference if the internal "references counter" of a slot referenced from this one already has zero value [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which is not "Owner" of the key slot	

⌋*(RS_CRYPTO_02009)*

8.21.1.3.3.8 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::GetPrototypedProps (SlotNumber *slotNum*, KeySlotPrototypeProps & *props*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_30118]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::GetPrototypedProps(SlotNumber slotNum, KeySlot PrototypeProps &props)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> GetPrototypedProps (SlotNumber slot Num, KeySlotPrototypeProps &props) const noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Parameters (out):	props	the output buffer for storing the prototype properties of the key slot
Return value:	ara::core::Result< void >	–

▽

△

Exception Safety:	noexcept
Thread Safety:	Thread-safe
Header file:	#include "ara/crypto/keys/key_storage_provider.h"
Description:	Get the prototyped properties of the key slot.
Notes:	[Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which has no any ("Owner" or "User") access rights to the key slot

⌋*(RS_CRYPTO_02110)*

8.21.1.3.3.9 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::GetContentProps (SlotNumber *slotNum*, KeySlotContentProps & *props*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_30119]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::GetContentProps(SlotNumber slotNum, KeySlotContent Props &props)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> GetContentProps (SlotNumber slotNum, KeySlotContentProps &props) const noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Parameters (out):	props	the output buffer for storing an actual properties of a content in the key slot
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Get an actual properties of a content in the key slot.	
Notes:	If this method called by a "User" Actor then always: props.exportability == false. [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kEmptyContainer if the slot is empty [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which has no any ("Owner" or "User") access rights to the key slot	

⌋*(RS_CRYPTO_02311)*

8.21.1.3.3.10 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::GetDefaultCryptoProviderUid (SlotNumber *slotNum,* CryptoProviderUid & *providerUid*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30120]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::GetDefaultCryptoProviderUid(SlotNumber slotNum, CryptoProviderUid &providerUid)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> GetDefaultCryptoProviderUid (SlotNumber slotNum, CryptoProviderUid &providerUid) const noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Parameters (out):	providerUid	the UID of Crypto Provider responsible for servicing of the slot
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Get UID of the default Crypto Provider assigned for servicing of the specified key slot.	
Notes:	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. If an object loaded to the slot can be loaded by any installed Crypto Provider then nil UID should be returned. This method checks the slot prototype meta-information ([Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which has no any ("Owner" or "User") access rights to the key slot	

⌋*(RS_CRYPTO_02401)*

8.21.1.3.3.11 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::GetOwner (SlotNumber *slotNum,* ActorUid & *ownerUid*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30121]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::keys::KeyStorageProvider::GetOwner(SlotNumber slotNum, ActorUid &ownerUid)
Scope:	class ara::crypto::keys::KeyStorageProvider

▽

△

Syntax:	`virtual ara::core::Result<void> GetOwner (SlotNumber slotNum, ActorUid &ownerUid) const noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Parameters (out):	ownerUid	the output buffer for storing the Owner UID of the key slot
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Get UID of an Actor granted by the "Owner" rights for the key slot.	
Notes:	[Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated)	

⌋*(RS_CRYPTO_02009)*

8.21.1.3.3.12 virtual ara::core::Result< std::size_t > ara::crypto::keys::KeyStorageProvider::GetUsers (SlotNumber *slotNum,* ara::core::Vector< UserPermissions > ∗ *users* = `nullptr`) const `[pure virtual], [noexcept]`

[SWS_CRYPT_30122]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::GetUsers(SlotNumber slotNum, ara::core::Vector< UserPermissions > *users=nullptr)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<std::size_t> GetUsers (SlotNumber slotNum, ara::core::Vector< UserPermissions > *users=nullptr) const noexcept=0;`	
Parameters (in):	slotNum	the target slot number
Parameters (out):	users	the optional pointer to a vector for storing output list
Return value:	ara::core::Result< std::size_t >	number of "Users" in the output list
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Get Users' Permissions list of all Actors granted by the User rights for the key slot.	
Notes:	If (users != nullptr) then capacity of the output vector should be enough for storing permissions of all "Users". [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the output vector is not enough	

⌋*(RS_CRYPTO_02009, RS_CRYPTO_02114)*

8.21.1.3.3.13 virtual ara::core::Result<TransactionId> ara::crypto::keys::KeyStorageProvider::BeginTransaction (const TransactionScope & *targetSlots*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30123]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::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 slots that should be updated during this transaction
Return value:	ara::core::Result< TransactionId >	an ID assigned to this transaction and unique on this ECU
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Begin new transaction for key slots update. A transaction is dedicated for updating related key slots simultaneously (in the atomic way). Each transaction must start from definition of its "scope" presented by a list of target slots that should be update during the transaction.	
Notes:	Whole code implementing single transaction must be located in a single thread! Any changes of the slots covered by a transaction become visible for "User" applications only after commit, as opposed to a change of a slot executed out of a transaction scope that become visible for "User" applications at once! Any Key Storage implementation should reserve double space for each key slot: 1st for "actual" (in-use) content and 2nd for "hidden" (new) content, updated during a transaction. [Error]: SecurityErrorDomain::kAccessViolation if targetSlots list has slot numbers that are not owned by current application [Error]: SecurityErrorDomain::kBusyResource if targetSlots list has slot numbers that are already involved to another pending transaction [Error]: SecurityErrorDomain::kInvalidArgument if targetSlots list has repetitions of slot numbers	

⌋*(RS_CRYPTO_02004)*

8.21.1.3.3.14 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::CommitTransaction (TransactionId *id*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30124]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::keys::KeyStorageProvider::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	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Commit changes of the transaction to Key Storage. The commit command permanently saves all changes made during the transaction in Key Storage and makes them visible to all "User" Actors/applications (according to their access rights).	
Notes:	Any changes of key slots made during a transaction are invisible to all "Users" (including the "Owner" application) up to the commit execution! [Error]: SecurityErrorDomain::kInvalidArgument if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back)	

⌋*(RS_CRYPTO_02004)*

8.21.1.3.3.15 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::RollbackTransaction (TransactionId *id*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30125]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::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	
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 "User" applications.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if provided id is invalid, i.e. this ID is unknown or correspondent transaction already was finished (commited or rolled back)	

⌋*(RS_CRYPTO_02004)*

8.21.1.3.3.16 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::UnsubscribeObserver (SlotNumber *slot*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30126]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::UnsubscribeObserver(SlotNumber slot)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> UnsubscribeObserver (SlotNumber 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	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Unsubscribe the Update Observer from changes monitoring of the specified slot.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if the specified slot is not monitored now (i.e. if it was not successfully opened via OpenAsUser() or it was already unsubscribed by this method)	

⌋*(RS_CRYPTO_02004)*

8.21.1.3.3.17 virtual SlotNumber ara::crypto::keys::KeyStorageProvider::FindReferringSlot (SlotNumber *targetSlot*, SlotNumber *previousFound* = kInvalidSlot) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30127]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::FindReferringSlot(SlotNumber targetSlot, SlotNumber previousFound=kInvalidSlot)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual SlotNumber FindReferringSlot (SlotNumber targetSlot, SlotNumber previousFound=kInvalidSlot) const noexcept=0;`	
Parameters (in):	targetSlot	the number of the target physical slot
	previousFound	the number of previous found key slot (the search will start from next slot number)
Return value:	SlotNumber	a number of found slot or kInvalidSlot if a slot referring to the target one was not found
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Find next slot that refers to the target one (due to the context dependency).	
Notes:	Use the previousFound = kInvalidSlot to start the search from the begin. If an application needs to find all slots referring to the target one then this method should be called multiple times until it will return kInvalidSlot. Also see cryp::CryptoObject::Save() for additional information about the referrences between slots.	

⌋*(RS_CRYPTO_02405)*

8.21.1.3.3.18 virtual ara::core::Result<void> ara::crypto::keys::KeyStorageProvider::ResetReference (SlotNumber *referrerSlot*, SlotNumber *referencedSlot* = kInvalidSlot) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30128]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::ResetReference(SlotNumber referrerSlot, SlotNumber referencedSlot=kInvalidSlot)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<void> ResetReference (SlotNumber referrerSlot, SlotNumber referencedSlot=kInvalidSlot) const noexcept=0;`	
Parameters (in):	referrerSlot	the number of the "referrer" (source) slot
	referencedSlot	the number of the "referenced" (target) slot
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Reset the reference from specified slot to another one (without the slot opening).	
Notes:	This operation can be executed only if the caller executable is "Owner" of the both slots. [Error]: SecurityErrorDomain::kUnreservedResource if the referrerSlot is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kBadObjectReference if an object in the the "referenced" slot has COUID different from the field KeySlotContentProps::mDependencyUid value of the "referrer" slot or if the objects in these slots has incompatible for referencing types [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which is not "Owner" of the both key slots	

⌋*(RS_CRYPTO_02405)*

8.21.1.3.3.19 virtual ara::core::Result<bool> ara::crypto::keys::KeyStorageProvider::CanLoadToCryptoProvider (SlotNumber *slotNum*, const CryptoProviderUid & *providerUid*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30129]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::CanLoadToCryptoProvider(SlotNumber slotNum, const CryptoProviderUid &providerUid)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual ara::core::Result<bool> CanLoadToCryptoProvider (SlotNumber slotNum, const CryptoProviderUid &providerUid) const noexcept=0;`	
Parameters (in):	slotNum	a slot number for the check

▽

△

	providerUid	an UID of Crypto Provider for the check
Return value:	ara::core::Result< bool >	true if the slot content can be loaded to specified provider and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Check the possibility to load an object from specified key slot to specified Crypto Provider.	
Notes:	If the providerUid has nil value then the method checks the possibility to load the slot content to any Crypto Provider installed in the system. This method should check not only the Crypto Providers "trust relationship matrix", but also actual content of the slot (object type, algorithm identifier, reference to dependency object) and the technical capabilities of the target Crypto Provider. [Error]: SecurityErrorDomain::kUnreservedResource if the slot number is incorrect (the slot is not allocated) [Error]: SecurityErrorDomain::kEmptyContainer if the slot is empty, but its prototype is not strict, i.e. if there is impossible to make the check due to lack of meta-information [Error]: SecurityErrorDomain::kAccessViolation if this method is called by an Actor, which has no any ("Owner" or "User") access rights to the key slot	

⌋*(RS_CRYPTO_02401)*

8.21.1.3.3.20 virtual UpdatesObserver::Sptr ara::crypto::keys::KeyStorageProvider::RegisterObserver (UpdatesObserver::Sptr *observer = nullptr*) [pure virtual], [noexcept]

[SWS_CRYPT_30130]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::RegisterObserver(UpdatesObserver::Sptr observer=nullptr)	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual UpdatesObserver::Sptr RegisterObserver (UpdatesObserver::Sptr 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::Sptr	shared 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.	

▽

△

Notes:	Only one instance of the UpdatesObserver may be registered by an application process, therefore this method always unregister previous observer and return its shared 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*)

8.21.1.3.3.21 virtual UpdatesObserver::Sptr ara::crypto::keys::KeyStorageProvider::GetRegisteredObserver ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30131]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::KeyStorageProvider::GetRegisteredObserver()	
Scope:	class ara::crypto::keys::KeyStorageProvider	
Syntax:	`virtual UpdatesObserver::Sptr GetRegisteredObserver () const noexcept=0;`	
Return value:	UpdatesObserver::Sptr	shared pointer to the registered Updates Observer interface (copy of an internal shared 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.	
Notes:	The method returns nullptr if no observers have been registered yet!	

⌋(*RS_CRYPTO_02401*)

8.21.1.3.3.22 virtual ara::core::Result<SlotNumber> ara::crypto::keys::KeyStorageProvider::FindSlot (const ara::core::InstanceSpecifier & *slotSpecifier*, CryptoProviderUid & *providerUid*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_30132]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::keys::KeyStorageProvider::FindSlot(const ara::core::InstanceSpecifier &slot Specifier, CryptoProviderUid &providerUid)
Scope:	class ara::crypto::keys::KeyStorageProvider

▽

△

Syntax:	`virtual ara::core::Result<SlotNumber> FindSlot (const ara::core::InstanceSpecifier &slotSpecifier, CryptoProviderUid &providerUid) const noexcept=0;`	
Parameters (in):	slotSpecifier	instance specifier of the target logical slot for search
Parameters (out):	providerUid	the UID of Crypto Provider responsible for servicing of the slot
Return value:	ara::core::Result< SlotNumber >	number of found slot or kInvalidSlot if a slot with such instance specifier was not found
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/key_storage_provider.h"	
Description:	Find a slot number by the instance specifier of the target logical slot.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if the slotSpecifier has incorrect value	

⌋*(RS_CRYPTO_02405)*

8.21.1.4 class ara::crypto::keys::UpdatesObserver

[SWS_CRYPT_30200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::keys::UpdatesObserver
Scope:	namespace ara::crypto::keys
Base class:	std::enable_shared_from_this< UpdatesObserver >
Syntax:	`class UpdatesObserver : public enable_shared_from_this< 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)*

Inherits enable_shared_from_this< UpdatesObserver >.

Public Types

- using Sptr = std::shared_ptr< UpdatesObserver >

Public Member Functions

- virtual ~UpdatesObserver () noexcept=default
- virtual void OnUpdate (const TransactionScope &updatedSlots) noexcept=0

8.21.1.4.1 Member Typedef Documentation

8.21.1.4.1.1 using ara::crypto::keys::UpdatesObserver::Sptr = std::shared_ptr<UpdatesObserver>

[SWS_CRYPT_30201]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::UpdatesObserver::Sptr
Scope:	class ara::crypto::keys::UpdatesObserver
Derived from:	std::shared_ptr<UpdatesObserver>
Syntax:	`using ara::crypto::keys::UpdatesObserver::Sptr = std::shared_ptr<UpdatesObserver>;`
Header file:	#include "ara/crypto/keys/updates_observer.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.21.1.4.2 Constructor & Destructor Documentation

8.21.1.4.2.1 virtual ara::crypto::keys::UpdatesObserver::~UpdatesObserver ( ) `[virtual]`, `[default]`, `[noexcept]`

[SWS_CRYPT_30210]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::keys::UpdatesObserver::~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_02311)*

8.21.1.5 struct ara::crypto::keys::UserPermissions

[SWS_CRYPT_30400]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::crypto::keys::UserPermissions
Scope:	namespace ara::crypto::keys
Syntax:	`struct UserPermissions {...};`
Header file:	#include "ara/crypto/keys/user_permissions.h"
Description:	Key slot User's Permissions prototype defined at the Design (or the Integration) stage.
Notes:	"Actor" is a permanently identifiable process defined by the Startup Configuration of an Executable. Access control management is based on the key slot attributes: (1) Each persistent slot always has only one "Owner Actor" that can save an object to the slot, clear it or copy its content to another owned slot. "Owner" is responsible for consistency the slot content. Only "Owner Actor" can execute export operation of a crypto object (if it is allowed by the object attributes). Owner's prototype defines the whole set of the allowed usage flags for owned key slot. (2) A "User" access right for a slot can be granted to an "Actor" by the Owner's manifest. The "User" access means the right to load a crypto object from the slot to a Crypto Provider's realm via the trusted container interface. Usage permissions of each "User" may be restricted independently from other. Additionally all "User Actors" obtain the "Exportability" attribute enforced to false, i.e. they cannot export objects independently from the actual attribute value (visible to "Owner"). (3) In order to have possibility load/use a key slot content, the "Owner" application also must have the "User" entry in the permissions table.

⌋*(RS_CRYPTO_02009, RS_CRYPTO_02114, RS_CRYPTO_02404)*

Public Attributes

- ActorUid mActorUid
- AllowedUsageFlags mAllowedUsage

8.21.1.5.1 Member Data Documentation

8.21.1.5.1.1 ActorUid ara::crypto::keys::UserPermissions::mActorUid

[SWS_CRYPT_30401]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::UserPermissions::mActorUid
Scope:	struct ara::crypto::keys::UserPermissions
Type:	ActorUid
Syntax:	`ActorUid ara::crypto::keys::UserPermissions::mActorUid;`
Header file:	#include "ara/crypto/keys/user_permissions.h"
Description:	UID of a "User Actor". "User Actor" has rights to load the crypto object to suitable crypto contexts.

⌋*(RS_CRYPTO_02114)*

8.21.1.5.1.2 AllowedUsageFlags ara::crypto::keys::UserPermissions::mAllowedUsage

[SWS_CRYPT_30402]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::UserPermissions::mAllowedUsage
Scope:	struct ara::crypto::keys::UserPermissions
Type:	AllowedUsageFlags
Syntax:	`AllowedUsageFlags ara::crypto::keys::UserPermissions::mAllowedUsage;`
Header file:	#include "ara/crypto/keys/user_permissions.h"
Description:	Restriction flags of allowed usage of a key stored to the slot for a process identified by the m ActorUid.

⌋*(RS_CRYPTO_02114)*

8.21.2 Typedef Documentation

8.21.2.1 using ara::crypto::keys::SlotNumber = typedef std::size_t

[SWS_CRYPT_30001]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::SlotNumber
Scope:	namespace ara::crypto::keys
Derived from:	typedef std::size_t
Syntax:	`using ara::crypto::keys::SlotNumber = std::size_t;`
Header file:	#include "ara/crypto/keys/elementary_types.h"
Description:	The Slot Number is intended for direct addressing of the "Key Slots" in the Key Storage.
Notes:	Each "Key Slot" is intended for storing of a single cryptographic object. Bit-length of this type is 32 or 64 bit depending from the target platform!

⌋*(RS_CRYPTO_02404)*

8.21.2.2 using ara::crypto::keys::TransactionId = typedef std::uint64_t

[SWS_CRYPT_30003]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::TransactionId
Scope:	namespace ara::crypto::keys

▽

△

Derived from:	typedef std::uint64_t
Syntax:	`using ara::crypto::keys::TransactionId = std::uint64_t;`
Header file:	#include "ara/crypto/keys/elementary_types.h"
Description:	Definition of a transaction identifier type.
Notes:	The zero value should be reserved for especial cases.

⌋(*RS_CRYPTO_02004*)

8.21.2.3 using ara::crypto::keys::TransactionScope = typedef ara::core::Vector<SlotNumber>

[SWS_CRYPT_30004]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::keys::TransactionScope
Scope:	namespace ara::crypto::keys
Derived from:	typedef ara::core::Vector<SlotNumber>
Syntax:	`using ara::crypto::keys::TransactionScope = ara::core::Vector<SlotNumber>;`
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*)

8.21.3 Enumeration Type Documentation

8.21.3.1 enum ara::crypto::keys::VersionControlType : std::uint8_t `[strong]`

[SWS_CRYPT_30005]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::keys::VersionControlType	
Scope:	namespace ara::crypto::keys	
Values:	kNone= 0	Version control is not applied for content of this slot.
	kLocal= 1	Version control is applied, slot initialization is not required, but only locally produced crypto object can be saved to the slot.
	kExternal= 2	Version control is applied, slot must be initialized by a COUID, specifying concrete external source of objects and minimal version.

▽

△

	kSwitchToLocal= 3	Similar to kLocal, but the slot can be initialized by an externally produced crypto object, which doesn't leave version control "track".
Header file:	#include "ara/crypto/keys/elementary_types.h"	
Description:	Enumeration of all Version Control Types.	
Notes:	Storage type: 8 bit unsigned integer.	

⌋*(RS_CRYPTO_02116)*

8.21.4 Function Documentation

8.21.4.1 ara::core::Result<KeyStorageProvider::Sptr> ara::crypto::keys::LoadKeyStorageProvider ( ) [noexcept]

[SWS_CRYPT_30099]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::LoadKeyStorageProvider()	
Scope:	namespace ara::crypto::keys	
Syntax:	`ara::core::Result<KeyStorageProvider::Sptr> LoadKeyStorageProvider () noexcept;`	
Return value:	ara::core::Result< KeyStorage Provider::Sptr >	shared smart pointer to loaded Key Storage Provider
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/keys/entry_point.h"	
Description:	Factory that creates or return existing single instance of the Key Storage Provider.	
Notes:	[Error]: SecurityErrorDomain::kRuntimeFault if the Key Storage Provider instance cannot be created	

⌋*(RS_CRYPTO_02109, RS_CRYPTO_02401)*

8.21.4.2 constexpr bool ara::crypto::keys::operator== (const KeySlotContentProps & *lhs*, const KeySlotContentProps & *rhs*) [noexcept]

[SWS_CRYPT_30550]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::keys::operator==(const KeySlotContentProps &lhs, const KeySlotContentProps &rhs)
Scope:	namespace ara::crypto::keys

▽

△

Syntax:	`inline 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, RS_CRYPTO_02311)*

8.21.4.3 constexpr bool ara::crypto::keys::operator!= (const KeySlotContentProps & *lhs,* const KeySlotContentProps & *rhs*) `[noexcept]`

[SWS_CRYPT_30551]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::operator!=(const KeySlotContentProps &lhs, const KeySlotContentProps &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`inline 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, RS_CRYPTO_02311)*

8.21.4.4 constexpr bool ara::crypto::keys::operator== (const KeySlotPrototypeProps & *lhs,* const KeySlotPrototypeProps & *rhs*) `[noexcept]`

[SWS_CRYPT_30350]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::operator==(const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`inline 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, RS_CRYPTO_02311)*

8.21.4.5 constexpr bool ara::crypto::keys::operator!= (const KeySlotPrototypeProps & *lhs,* const KeySlotPrototypeProps & *rhs*) `[noexcept]`

[SWS_CRYPT_30351]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::operator!=(const KeySlotPrototypeProps &lhs, const KeySlotPrototypeProps &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`inline 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, RS_CRYPTO_02311)*

8.21.4.6 virtual void ara::crypto::keys::UpdatesObserver::OnUpdate (const TransactionScope & *updatedSlots*) `[pure virtual], [noexcept]`

[SWS_CRYPT_30211]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::UpdatesObserver::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.	
Notes:	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)*

8.21.4.7 constexpr bool ara::crypto::keys::operator== (const UserPermissions & *lhs*, const UserPermissions & *rhs*) [noexcept]

[SWS_CRYPT_30450]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::operator==(const UserPermissions &lhs, const UserPermissions &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`inline constexpr bool operator== (const UserPermissions &lhs, const UserPermissions &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/user_permissions.h"	
Description:	Comparison operator "equal" for UserPermissions operands.	

⌋*(RS_CRYPTO_02114, RS_CRYPTO_02311)*

8.21.4.8 constexpr bool ara::crypto::keys::operator!= (const UserPermissions & *lhs*, const UserPermissions & *rhs*) [noexcept]

[SWS_CRYPT_30451]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::keys::operator!=(const UserPermissions &lhs, const UserPermissions &rhs)	
Scope:	namespace ara::crypto::keys	
Syntax:	`inline constexpr bool operator!= (const UserPermissions &lhs, const UserPermissions &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/user_permissions.h"	
Description:	Comparison operator "not equal" for UserPermissions operands.	

⌋*(RS_CRYPTO_02114, RS_CRYPTO_02311)*

8.21.5 Variable Documentation

8.21.5.1 const SlotNumber ara::crypto::keys::kInvalidSlot = static_cast<SlotNumber>(-1LL)

[SWS_CRYPT_30002]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::keys::kInvalidSlot
Scope:	namespace ara::crypto::keys
Type:	const SlotNumber
Syntax:	`const SlotNumber ara::crypto::keys::kInvalidSlot= static_cast<SlotNumber>(-1LL);`
Header file:	#include "ara/crypto/keys/elementary_types.h"
Description:	A reserved slot number that cannot be used for addressing of real slots of the storage.

⌋*(RS_CRYPTO_02311)*

8.22 X.509 Provider API

Namespaces

- ara::crypto::x509

Classes

- class ara::crypto::x509::BasicCertInfo
- class ara::crypto::x509::CertSignRequest
- class ara::crypto::x509::Certificate
- class ara::crypto::x509::OcspRequest
- class ara::crypto::x509::OcspResponse
- class ara::crypto::x509::X509DN
- class ara::crypto::x509::X509Extensions
- class ara::crypto::x509::X509Provider

Enumerations

Functions

- ara::core::Result< X509Provider::Sptr > ara::crypto::x509::LoadX509Provider () noexcept

Detailed Description

X.509 Provider represents unified interface to whole client-side functionality of X.509 compliant Public Key Infrastructure (PKI). Crypto Stack exposes a single instance of X.509 Provider responsible for implementation of complete functionality of local certificates management, including certification requests preparation, certificates parsing and validation, trust management.

8.22.1 Class Documentation

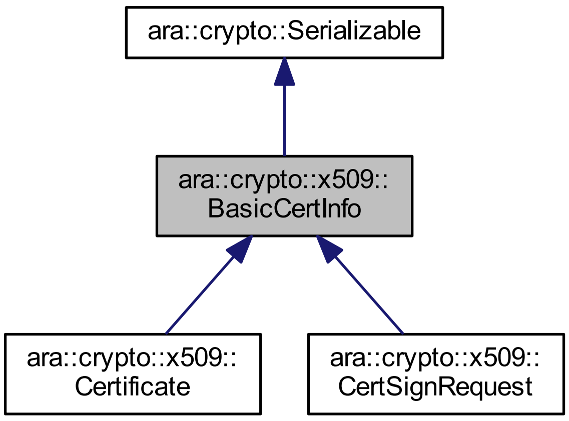
8.22.1.1 class ara::crypto::x509::BasicCertInfo

[SWS_CRYPT_40100]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::BasicCertInfo
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::Serializable
Syntax:	`class BasicCertInfo :  public Serializable {...};`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	Basic Certificate Information interface.

⌋*(RS_CRYPTO_02306)*

Inheritance diagram for ara::crypto::x509::BasicCertInfo:



Public Types

- using KeyConstraints = std::uint32_t

Public Member Functions

- virtual const cryp::X509PublicKeyInfo & SubjectPubKey (cryp::CryptoProvider::Sptr cryptoProvider=nullptr) const noexcept=0
- virtual const X509DN & SubjectDn () const noexcept=0
- virtual bool IsCa () const noexcept=0
- virtual std::uint32_t GetPathLimit () const noexcept=0
- virtual KeyConstraints GetConstraints () const noexcept=0

Constants of X.509 v3 Key Constraints

- static const KeyConstraints kConstrNone = 0
- static const KeyConstraints kConstrDigitalSignature = 0x08000
- static const KeyConstraints kConstrNonRepudiation = 0x04000
- static const KeyConstraints kConstrKeyEncipherment = 0x02000
- static const KeyConstraints kConstrDataEncipherment = 0x01000
- static const KeyConstraints kConstrKeyAgreement = 0x00800
- static const KeyConstraints kConstrKeyCertSign = 0x00400
- static const KeyConstraints kConstrCrlSign = 0x00200
- static const KeyConstraints kConstrEncipherOnly = 0x00100
- static const KeyConstraints kConstrDecipherOnly = 0x00080

Additional Inherited Members

8.22.1.1.1 Member Typedef Documentation

8.22.1.1.1.1 using ara::crypto::x509::BasicCertInfo::KeyConstraints = std::uint32_t

[SWS_CRYPT_40101]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::BasicCertInfo::KeyConstraints
Scope:	class ara::crypto::x509::BasicCertInfo
Derived from:	std::uint32_t
Syntax:	`using ara::crypto::x509::BasicCertInfo::KeyConstraints = std::uint32_t;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	X.509 v3 Key Constraints type definition.

⌋*(RS_CRYPTO_02311)*

8.22.1.1.2 Member Function Documentation

8.22.1.1.2.1 virtual const cryp::X509PublicKeyInfo& ara::crypto::x509::BasicCertInfo::SubjectPubKey (cryp::CryptoProvider::Sptr *cryptoProvider* = `nullptr`) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40111]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::BasicCertInfo::SubjectPubKey(cryp::CryptoProvider::Sptr cryptoProvider=nullptr)	
Scope:	class ara::crypto::x509::BasicCertInfo	
Syntax:	`virtual const cryp::X509PublicKeyInfo& SubjectPubKey (cryp::CryptoProvider::Sptr cryptoProvider=nullptr) const noexcept=0;`	
Parameters (in):	cryptoProvider	shared pointer of a target Crypto Provider, where the public key will be used
Return value:	const cryp::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.	
Notes:	If (cryptoProvider == nullptr) then X509PublicKeyInfo object will be loaded in realm of the Stack-default Crypto Provider	

⌋*(RS_CRYPTO_02311)*

8.22.1.1.2.2 virtual const X509DN& ara::crypto::x509::BasicCertInfo::SubjectDn ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40112]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::BasicCertInfo::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_02311)*

8.22.1.1.2.3 virtual bool ara::crypto::x509::BasicCertInfo::IsCa ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40113]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::BasicCertInfo::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:	Find out whether this is a CA request.	

⌋*(RS_CRYPTO_02311)*

8.22.1.1.2.4 virtual std::uint32_t ara::crypto::x509::BasicCertInfo::GetPathLimit ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40114]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::BasicCertInfo::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_02311)*

8.22.1.1.2.5 virtual KeyConstraints ara::crypto::x509::BasicCertInfo::GetConstraints ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40115]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::BasicCertInfo::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_02311)*

8.22.1.1.3 Member Data Documentation

8.22.1.1.3.1 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrNone = 0 [static]

[SWS_CRYPT_40150]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrNone
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrNone= 0;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	No key constraints.

⌋*(RS_CRYPTO_02311)*

8.22.1.1.3.2 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDigitalSignature = 0x08000 [static]

[SWS_CRYPT_40151]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrDigitalSignature
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDigitalSignature= 0x08000;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for digital signature production.

⌋*(RS_CRYPTO_02311)*

8.22.1.1.3.3 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrNonRepudiation = 0x04000 [static]

[SWS_CRYPT_40152]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrNonRepudiation
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrNonRepudiation= 0x04000;`
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_02311)*

8.22.1.1.3.4 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyEncipherment = 0x02000 [static]

[SWS_CRYPT_40153]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrKeyEncipherment
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyEncipherment= 0x02000;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for key encipherment.

⌋*(RS_CRYPTO_02311)*

8.22.1.1.3.5 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDataEncipherment = 0x01000 [static]

[SWS_CRYPT_40154]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrDataEncipherment
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDataEncipherment= 0x01000;`

▽

△

Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for data encipherment.

⌋(*RS_CRYPTO_02311*)

8.22.1.1.3.6 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyAgreement = 0x00800 `[static]`

[SWS_CRYPT_40155]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrKeyAgreement
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyAgreement= 0x00800;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for a key agreement protocol execution.

⌋(*RS_CRYPTO_02311*)

8.22.1.1.3.7 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyCertSign = 0x00400 `[static]`

[SWS_CRYPT_40156]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrKeyCertSign
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrKeyCertSign= 0x00400;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for certificates signing.

⌋(*RS_CRYPTO_02311*)

8.22.1.1.3.8 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrCrlSign = 0x00200 `[static]`

[SWS_CRYPT_40157]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrCrlSign
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrCrlSign= 0x00200;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The key can be used for Certificates Revokation Lists (CRL) signing.

⌋*(RS_CRYPTO_02311)*

8.22.1.1.3.9 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrEncipherOnly = 0x00100 `[static]`

[SWS_CRYPT_40158]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrEncipherOnly
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrEncipherOnly= 0x00100;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The enciphermet key can be used for enciphering only.

⌋*(RS_CRYPTO_02311)*

8.22.1.1.3.10 const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDecipherOnly = 0x00080 `[static]`

[SWS_CRYPT_40159]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::BasicCertInfo::kConstrDecipherOnly
Scope:	class ara::crypto::x509::BasicCertInfo
Type:	const KeyConstraints
Syntax:	`const KeyConstraints ara::crypto::x509::BasicCertInfo::kConstrDecipherOnly= 0x00080;`
Header file:	#include "ara/crypto/x509/basic_cert_info.h"
Description:	The enciphermet key can be used for deciphering only.

⌋*(RS_CRYPTO_02311)*

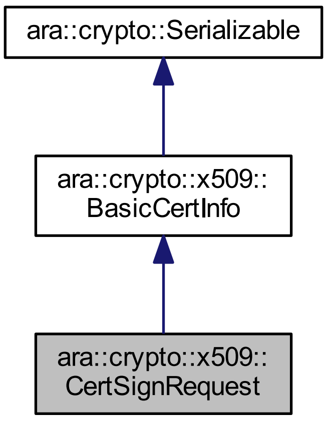
8.22.1.2 class ara::crypto::x509::CertSignRequest

[SWS_CRYPT_40300]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::CertSignRequest
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::x509::BasicCertInfo
Syntax:	`class CertSignRequest : public BasicCertInfo {...};`
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Description:	Certificate Signing Request (CSR) object interface
Notes:	This interface is dedicated for complete parsing of the request content.

⌋*(RS_CRYPTO_02306)*

Inheritance diagram for ara::crypto::x509::CertSignRequest:


ara::crypto::Serializable
ara::crypto::x509:: BasicCertInfo
ara::crypto::x509:: CertSignRequest

Public Types

- using Uptrc = std::unique_ptr< const CertSignRequest, CustomDeleter >
- using Uptr = std::unique_ptr< CertSignRequest, CustomDeleter >

Public Member Functions

- virtual bool Verify () const noexcept=0
- virtual ara::core::Result< std::size_t > ChallengePassword (ara::core::String ∗password=nullptr) const noexcept=0

Additional Inherited Members

8.22.1.2.1 Member Typedef Documentation

8.22.1.2.1.1 using ara::crypto::x509::CertSignRequest::Uptrc = std::unique_ptr<const CertSignRequest, CustomDeleter>

[SWS_CRYPT_40301]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::CertSignRequest::Uptrc
Scope:	class ara::crypto::x509::CertSignRequest
Derived from:	std::unique_ptr<const CertSignRequest, CustomDeleter>
Syntax:	`using ara::crypto::x509::CertSignRequest::Uptrc = std::unique_ptr<const CertSignRequest, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Description:	Unique smart pointer of the constant interface.

⌋*(RS_CRYPTO_02404)*

8.22.1.2.1.2 using ara::crypto::x509::CertSignRequest::Uptr = std::unique_ptr<CertSignRequest, CustomDeleter>

[SWS_CRYPT_40302]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::CertSignRequest::Uptr
Scope:	class ara::crypto::x509::CertSignRequest
Derived from:	std::unique_ptr<CertSignRequest, CustomDeleter>
Syntax:	`using ara::crypto::x509::CertSignRequest::Uptr = std::unique_ptr<CertSignRequest, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/cert_sign_request.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.22.1.2.2 Member Function Documentation

8.22.1.2.2.1 virtual bool ara::crypto::x509::CertSignRequest::Verify ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40311]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::CertSignRequest::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_02311)*

8.22.1.2.2.2 virtual ara::core::Result<std::size_t> ara::crypto::x509::CertSignRequest::ChallengePassword (ara::core::String ∗ *password* = *nullptr*) const [pure virtual], [noexcept]

[SWS_CRYPT_40312]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::CertSignRequest::ChallengePassword(ara::core::String *password=nullptr)	
Scope:	class ara::crypto::x509::CertSignRequest	
Syntax:	`virtual ara::core::Result<std::size_t> ChallengePassword (ara::core::String *password=nullptr) const noexcept=0;`	
Parameters (out):	password	the optional pointer to an output string
Return value:	ara::core::Result< std::size_t >	length of the password if it was provided or 0 otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/cert_sign_request.h"	
Description:	Get the challenge password for this request (if it was included to the request).	
Notes:	[Error]: SecurityErrorDomain::kInsufficientCapacity if (password != nullptr), but its capacity is less then required for storing the password value	

⌋*(RS_CRYPTO_02311)*

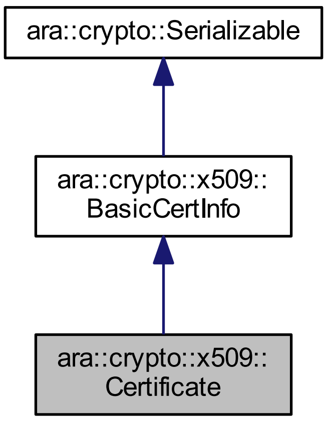
8.22.1.3 class ara::crypto::x509::Certificate

[SWS_CRYPT_40200]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::Certificate
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::x509::BasicCertInfo
Syntax:	`class Certificate :  public BasicCertInfo {...};`
Header file:	#include "ara/crypto/x509/certificate.h"
Description:	X.509 Certificate interface.

⌋*(RS_CRYPTO_02306)*

Inheritance diagram for ara::crypto::x509::Certificate:



Public Types

- using Uptr = std::unique_ptr< Certificate, CustomDeleter >
- using Uptrc = std::unique_ptr< const Certificate, CustomDeleter >

Public Member Functions

- virtual std::uint32_t X509Version () const noexcept=0
- virtual bool IsRoot () const noexcept=0
- virtual const X509DN & IssuerDn () const =0
- virtual time_t StartTime () const noexcept=0
- virtual time_t EndTime () const noexcept=0

- virtual ara::core::Result< std::size_t > SerialNumber (WritableMemRegion sn=WritableMemRegion()) const noexcept=0
- virtual ara::core::Result< std::size_t > AuthorityKeyId (WritableMemRegion id=WritableMemRegion()) const noexcept=0
- virtual ara::core::Result< std::size_t > SubjectKeyId (WritableMemRegion id=WritableMemRegion()) const noexcept=0
- virtual bool VerifyMe (const Certificate ∗caCert=nullptr) const noexcept=0
- virtual std::size_t GetFingerprint (ReadWriteMemRegion fingerprint, cryp::HashFunctionCtx &hashCtx) const noexcept=0
- virtual Status GetStatus () const noexcept=0

Additional Inherited Members

8.22.1.3.1 Member Typedef Documentation

8.22.1.3.1.1 using ara::crypto::x509::Certificate::Uptr = std::unique_ptr<Certificate, CustomDeleter>

[SWS_CRYPT_40201]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::Certificate::Uptr
Scope:	class ara::crypto::x509::Certificate
Derived from:	std::unique_ptr<Certificate, CustomDeleter>
Syntax:	`using ara::crypto::x509::Certificate::Uptr = std::unique_ptr<Certificate, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/certificate.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.22.1.3.1.2 using ara::crypto::x509::Certificate::Uptrc = std::unique_ptr<const Certificate, CustomDeleter>

[SWS_CRYPT_40202]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::Certificate::Uptrc
Scope:	class ara::crypto::x509::Certificate

▽

△

Derived from:	std::unique_ptr<const Certificate, CustomDeleter>
Syntax:	`using ara::crypto::x509::Certificate::Uptrc = std::unique_ptr<const Certificate, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/certificate.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.22.1.3.2 Member Enumeration Documentation

8.22.1.3.2.1 enum ara::crypto::x509::Certificate::Status : std::uint8_t `[strong]`

[SWS_CRYPT_40203]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::x509::Certificate::Status	
Scope:	class ara::crypto::x509::Certificate	
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.
	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.	
Notes:	Storage type: 8 bit unsigned integer.	

⌋*(RS_CRYPTO_02306)*

8.22.1.3.3 Member Function Documentation

8.22.1.3.3.1 virtual std::uint32_t ara::crypto::x509::Certificate::X509Version ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40211]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::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_02311)*

8.22.1.3.3.2 virtual bool ara::crypto::x509::Certificate::IsRoot ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40212]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::IsRoot()	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual bool IsRoot () const noexcept=0;`	
Return value:	bool	true if this certificate belongs to a root CA (i.e. the certificate is self-signed)
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_02311)*

8.22.1.3.3.3 virtual const X509DN& ara::crypto::x509::Certificate::IssuerDn ( ) const `[pure virtual]`

[SWS_CRYPT_40213]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::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_02311)*

8.22.1.3.3.4 virtual time_t ara::crypto::x509::Certificate::StartTime ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40214]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::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_02311)*

8.22.1.3.3.5 virtual time_t ara::crypto::x509::Certificate::EndTime ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40215]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::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_02311)*

8.22.1.3.3.6 virtual ara::core::Result<std::size_t> ara::crypto::x509::Certificate::SerialNumber (WritableMemRegion *sn* = WritableMemRegion*()*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40216]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::SerialNumber(WritableMemRegion sn=WritableMemRegion())	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual ara::core::Result<std::size_t> SerialNumber (WritableMemRegion sn=WritableMemRegion()) const noexcept=0;`	
Parameters (out):	sn	an optional output buffer for storing the serial number
Return value:	ara::core::Result< std::size_t >	size of the certificate serial number in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the serial number of this certificate.	
Notes:	If (sn.empty() == true) then this method only returns required size of the output buffer. [Error]: SecurityErrorDomain::kInsufficientCapacity if (sn.empty() == false), but its size is not enough for storing the output value	

⌋*(RS_CRYPTO_02311)*

8.22.1.3.3.7 virtual ara::core::Result<std::size_t> ara::crypto::x509::Certificate::AuthorityKeyId (WritableMemRegion *id* = WritableMemRegion*()*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40217]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::AuthorityKeyId(WritableMemRegion id=WritableMemRegion())	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual ara::core::Result<std::size_t> AuthorityKeyId (WritableMemRegion id=WritableMemRegion()) const noexcept=0;`	
Parameters (out):	id	the optional output buffer
Return value:	ara::core::Result< std::size_t >	size of the DER encoded AuthorityKeyIdentifier in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the DER encoded AuthorityKeyIdentifier of this certificate.	
Notes:	If (id.empty() == true) then this method only returns required size of the output buffer. [Error]: SecurityErrorDomain::kInsufficientCapacity if (id.empty() == false), but its size is not enough for storing the output value	

⌋*(RS_CRYPTO_02311)*

8.22.1.3.3.8 virtual ara::core::Result<std::size_t> ara::crypto::x509::Certificate::SubjectKeyId (WritableMemRegion *id* = WritableMemRegion*()*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40218]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::SubjectKeyId(WritableMemRegion id=WritableMemRegion())	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual ara::core::Result<std::size_t> SubjectKeyId (WritableMemRegion id=WritableMemRegion()) const noexcept=0;`	
Parameters (out):	id	the optional output buffer
Return value:	ara::core::Result< std::size_t >	size of the DER encoded SubjectKeyIdentifier in bytes
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Get the DER encoded SubjectKeyIdentifier of this certificate.	
Notes:	If (id.empty() == true) then this method only returns required size of the output buffer. [Error]: SecurityErrorDomain::kInsufficientCapacity if (id.empty() == false), but its size is not enough for storing the output value	

⌋*(RS_CRYPTO_02311)*

8.22.1.3.3.9 virtual bool ara::crypto::x509::Certificate::VerifyMe (const Certificate ∗ *caCert* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40219]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::VerifyMe(const Certificate *caCert=nullptr)	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual bool VerifyMe (const Certificate *caCert=nullptr) 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.	
Notes:	Call with (caCert == nullptr) is applicable only if this is a certificate of a root CA.	

⌋*(RS_CRYPTO_02311)*

8.22.1.3.3.10 virtual std::size_t ara::crypto::x509::Certificate::GetFingerprint (ReadWriteMemRegion *fingerprint*, cryp::HashFunctionCtx & *hashCtx*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40220]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::GetFingerprint(ReadWriteMemRegion fingerprint, cryp::HashFunctionCtx &hashCtx)	
Scope:	class ara::crypto::x509::Certificate	
Syntax:	`virtual 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:	std::size_t	number of bytes actually saved to the output buffer
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/certificate.h"	
Description:	Calculate a fingerprint from the whole certificate.	
Notes:	The produced fingerprint value saved to the output buffer starting from the least significant 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. [Error]: SecurityErrorDomain::kIncompleteArgState if the hashCtx context is not initialized by required domain parameters	

⌋*(RS_CRYPTO_02311)*

8.22.1.3.3.11 virtual Status ara::crypto::x509::Certificate::GetStatus ( ) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40221]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::Certificate::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_02311)*

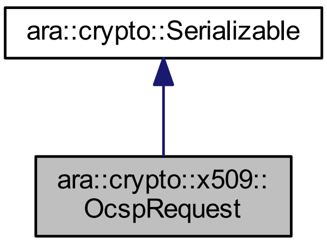
8.22.1.4 class ara::crypto::x509::OcspRequest

[SWS_CRYPT_40700]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::OcspRequest
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::Serializable
Syntax:	`class OcspRequest : public Serializable {...};`
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Description:	On-line Certificate Status Protocol Request.

⌋(*RS_CRYPTO_02306*)

Inheritance diagram for ara::crypto::x509::OcspRequest:



Public Types

- using Uptr = std::unique_ptr< OcspRequest, CustomDeleter >
- using Uptrc = std::unique_ptr< const OcspRequest, CustomDeleter >

Public Member Functions

- virtual std::uint32_t Version () const noexcept=0

Additional Inherited Members

8.22.1.4.1 Member Typedef Documentation

8.22.1.4.1.1 using ara::crypto::x509::OcspRequest::Uptr = std::unique_ptr<OcspRequest, CustomDeleter>

[SWS_CRYPT_40701]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::OcspRequest::Uptr
Scope:	class ara::crypto::x509::OcspRequest
Derived from:	std::unique_ptr<OcspRequest, CustomDeleter>
Syntax:	`using ara::crypto::x509::OcspRequest::Uptr = std::unique_ptr<OcspRequest, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.22.1.4.1.2 using ara::crypto::x509::OcspRequest::Uptrc = std::unique_ptr<const OcspRequest, CustomDeleter>

[SWS_CRYPT_40702]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::OcspRequest::Uptrc
Scope:	class ara::crypto::x509::OcspRequest
Derived from:	std::unique_ptr<const OcspRequest, CustomDeleter>
Syntax:	`using ara::crypto::x509::OcspRequest::Uptrc = std::unique_ptr<const OcspRequest, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/ocsp_request.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.22.1.4.2 Member Function Documentation

8.22.1.4.2.1 virtual std::uint32_t ara::crypto::x509::OcspRequest::Version ( ) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40711]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::OcspRequest::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_02311)*

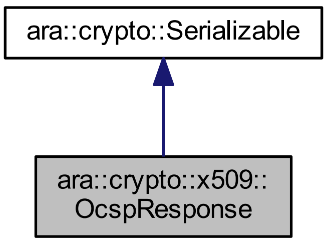
8.22.1.5 class ara::crypto::x509::OcspResponse

[SWS_CRYPT_40800]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::OcspResponse
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::Serializable
Syntax:	`class OcspResponse : public Serializable {...};`
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Description:	On-line Certificate Status Protocol Response.

⌋*(RS_CRYPTO_02306)*

Inheritance diagram for ara::crypto::x509::OcspResponse:



Public Types

- using Uptr = std::unique_ptr< OcspResponse, CustomDeleter >

- using Uptrc = std::unique_ptr< const OcspResponse, CustomDeleter >

Public Member Functions

- virtual std::uint32_t Version () const noexcept=0

Additional Inherited Members

8.22.1.5.1 Member Typedef Documentation

8.22.1.5.1.1 using ara::crypto::x509::OcspResponse::Uptr = std::unique_ptr<OcspResponse, CustomDeleter>

[SWS_CRYPT_40801]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::OcspResponse::Uptr
Scope:	class ara::crypto::x509::OcspResponse
Derived from:	std::unique_ptr<OcspResponse, CustomDeleter>
Syntax:	`using ara::crypto::x509::OcspResponse::Uptr = std::unique_ptr<OcspResponse, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.22.1.5.1.2 using ara::crypto::x509::OcspResponse::Uptrc = std::unique_ptr<const OcspResponse, CustomDeleter>

[SWS_CRYPT_40802]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::OcspResponse::Uptrc
Scope:	class ara::crypto::x509::OcspResponse
Derived from:	std::unique_ptr<const OcspResponse, CustomDeleter>
Syntax:	`using ara::crypto::x509::OcspResponse::Uptrc = std::unique_ptr<const OcspResponse, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/ocsp_response.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.22.1.5.2 Member Function Documentation

8.22.1.5.2.1 virtual std::uint32_t ara::crypto::x509::OcspResponse::Version ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_40811]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::OcspResponse::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_02311)*

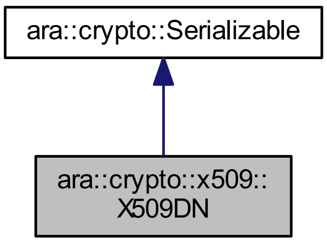
8.22.1.6 class ara::crypto::x509::X509DN

[SWS_CRYPT_40400]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::X509DN
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::Serializable
Syntax:	`class X509DN : public Serializable {...};`
Header file:	#include "ara/crypto/x509/x509_dn.h"
Description:	Interface of X.509 Distinguished Name (DN).

⌋*(RS_CRYPTO_02306)*

Inheritance diagram for ara::crypto::x509::X509DN:



Public Types

- using Uptr = std::unique_ptr< X509DN, CustomDeleter >
- using Uptrc = std::unique_ptr< const X509DN, CustomDeleter >

Public Member Functions

- virtual ara::core::Result< std::size_t > GetDnString (ara::core::String *dn=nullptr) const noexcept=0
- virtual ara::core::Result< void > SetDn (ara::core::StringView dn) noexcept=0
- virtual ara::core::Result< std::size_t > GetAttribute (AttributeId id, ara::core::String *attribute=nullptr) const noexcept=0
- virtual ara::core::Result< void > SetAttribute (AttributeId id, ara::core::StringView attribute) const noexcept=0
- virtual ara::core::Result< std::size_t > GetAttribute (AttributeId id, unsigned index, ara::core::String *attribute=nullptr) const noexcept=0
- virtual ara::core::Result< void > SetAttribute (AttributeId id, unsigned index, ara::core::StringView attribute) const noexcept=0

Additional Inherited Members

8.22.1.6.1 Member Typedef Documentation

8.22.1.6.1.1 using ara::crypto::x509::X509DN::Uptr = std::unique_ptr<X509DN, CustomDeleter>

[SWS_CRYPT_40401]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::X509DN::Uptr
Scope:	class ara::crypto::x509::X509DN
Derived from:	std::unique_ptr<X509DN, CustomDeleter>
Syntax:	`using ara::crypto::x509::X509DN::Uptr = std::unique_ptr<X509DN, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/x509_dn.h"
Description:	Unique smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.22.1.6.1.2 using ara::crypto::x509::X509DN::Uptrc = std::unique_ptr<const X509DN, CustomDeleter>

[SWS_CRYPT_40402]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::X509DN::Uptrc
Scope:	class ara::crypto::x509::X509DN
Derived from:	std::unique_ptr<const X509DN, CustomDeleter>
Syntax:	`using ara::crypto::x509::X509DN::Uptrc = std::unique_ptr<const X509DN, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/x509_dn.h"
Description:	Unique smart pointer of the constant interface.

⌋*(RS_CRYPTO_02404)*

8.22.1.6.2 Member Enumeration Documentation

8.22.1.6.2.1 enum ara::crypto::x509::X509DN::AttributeId : std::uint8_t `[strong]`

[SWS_CRYPT_40403]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::x509::X509DN::AttributeId	
Scope:	class ara::crypto::x509::X509DN	
Values:	kCommonName= 0	Common Name.
	kCountry= 1	Country.
	kState= 2	State.
	kLocality= 3	Locality.
	kOrganization= 4	Organization.
	kOrgUnit= 5	Organization Unit.
	kStreet= 6	Street.
	kPostalCode= 7	Postal Code.
	kTitle= 8	Title.
	kSurname= 9	Surname.
	kGivenName= 10	Given Name.
	kInitials= 11	Initials.
	kPseudonym= 12	Pseudonym.
	kGenerationQualifier= 13	Generation Qualifier.
	kDomainComponent= 14	Domain Component.
	kDnQualifier= 15	Distinguished Name Qualifier.
	kEmail= 16	E-mail.
	kUri= 17	URI.

▽

△

	kDns= 18	DNS.
	kHostName= 19	Host Name (UNSTRUCTUREDNAME)
	kIpAddress= 20	IP Address (UNSTRUCTUREDADDRESS)
	kSerialNumbers= 21	Serial Numbers.
	kUserId= 22	User ID.
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Enumeration of DN attributes' identifiers.	
Notes:	Storage type: 8 bit unsigned integer.	

⌋*(RS_CRYPTO_02311)*

8.22.1.6.3 Member Function Documentation

8.22.1.6.3.1 virtual ara::core::Result<std::size_t> ara::crypto::x509::X509DN::GetDnString (ara::core::String ∗ *dn* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509DN::GetDnString(ara::core::String *dn=nullptr)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<std::size_t> GetDnString (ara::core::String *dn=nullptr) const noexcept=0;`	
Parameters (out):	dn	the pointer to a string for storing whole DN value as a single string
Return value:	ara::core::Result< std::size_t >	length of the whole DN string
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Get the whole Distinguished Name (DN) as a single string.	
Notes:	Capacity of the output string must be enough for storing the output value! If (dn == nullptr) then method only returns required buffer capacity. [Error]: SecurityErrorDomain::kInsufficientCapacity if (dn != nullptr), but dn->capacity() is less than required for the output value storing	

⌋*(RS_CRYPTO_02311)*

8.22.1.6.3.2 virtual ara::core::Result<void> ara::crypto::x509::X509DN::SetDn (ara::core::StringView *dn*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40412]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509DN::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.	
Notes:	[Error]: SecurityErrorDomain::kUnexpectedValue if the dn string has incorrect syntax	

⌋(*RS_CRYPTO_02311*)

8.22.1.6.3.3 virtual ara::core::Result$<$std::size_t$>$ ara::crypto::x509::X509DN::GetAttribute (AttributeId *id*, ara::core::String $*$ *attribute* = `nullptr`) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40413]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509DN::GetAttribute(AttributeId id, ara::core::String *attribute=nullptr)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<std::size_t> GetAttribute (AttributeId id, ara::core::String *attribute=nullptr) const noexcept=0;`	
Parameters (in):	id	the identifier of required attribute
Parameters (out):	attribute	the pointer to a string for storing attribute value
Return value:	ara::core::Result< std::size_t >	length of the attribute value (0 for empty attributes)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Get DN attribute by its ID (this method is applicale to all attributes except kOrgUnit and k DomainComponent).	
Notes:	Capacity of the output string must be enough for storing the output value! If (attribute == nullptr) then method only returns required buffer capacity. [Error]: SecurityErrorDomain::kUnknownIdentifier if the id argument has unsupported value [Error]: SecurityErrorDomain::kInsufficientCapacity if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output	

⌋(*RS_CRYPTO_02311*)

8.22.1.6.3.4 virtual ara::core::Result<void> ara::crypto::x509::X509DN::SetAttribute (AttributeId *id,* ara::core::StringView *attribute*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40414]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509DN::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	
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).	
Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if the id argument has unsupported value [Error]: SecurityErrorDomain::kUnexpectedValue if the attribute string contains incorrect characters or it has unsupported length	

⌋*(RS_CRYPTO_02311)*

8.22.1.6.3.5 virtual ara::core::Result<std::size_t> ara::crypto::x509::X509DN::GetAttribute (AttributeId *id,* unsigned *index,* ara::core::String ∗ *attribute* = `nullptr`) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40415]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509DN::GetAttribute(AttributeId id, unsigned index, ara::core::String *attribute=nullptr)	
Scope:	class ara::crypto::x509::X509DN	
Syntax:	`virtual ara::core::Result<std::size_t> GetAttribute (AttributeId id, unsigned index, ara::core::String *attribute=nullptr) const noexcept=0;`	
Parameters (in):	id	the identifier of required attribute
	index	the zero-based index of required component of the attribute
Parameters (out):	attribute	the pointer to a string for storing attribute value
Return value:	ara::core::Result< std::size_t >	length of the attribute value (0 for empty attributes)
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	

▽

△

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).
Notes:	Capacity of the output string must be enough for storing the output value! If (attribute == nullptr) then method only returns required buffer capacity. [Error]: SecurityErrorDomain::kUnknownIdentifier if the id argument has unsupported value [Error]: SecurityErrorDomain::kInsufficientCapacity if (attribute != nullptr), but attribute->capacity() is less than required for storing of the output [Error]: SecurityErrorDomain::kInvalidArgument if (id != kOrgUnit) && (id != kDomain Component) && (index > 0) [Error]: SecurityErrorDomain::kAboveBoundary if ((id == kOrgUnit) || (id == kDomain Component)) and the index value is greater than or equal to the actual number of components in the specified attribute

⌋(*RS_CRYPTO_02311*)

8.22.1.6.3.6 virtual ara::core::Result<void> ara::crypto::x509::X509DN::SetAttribute (AttributeId *id*, unsigned *index*, ara::core::StringView *attribute*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40416]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509DN::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;`	
Parameters (in):	id	the identifier of required attribute
	index	the zero-based index of required component of the attribute
	attribute	the attribute value
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_dn.h"	
Description:	Set DN attribute by its ID and sequential index (this method is applicale to attributes kOrgUnit and kDomainComponent).	

▽

△

Notes:	[Error]: SecurityErrorDomain::kUnknownIdentifier if the id argument has unsupported value [Error]: SecurityErrorDomain::kUnexpectedValue if the attribute string contains incorrect characters or it has unsupported length [Error]: SecurityErrorDomain::kInvalidArgument if (id != kOrgUnit) && (id != kDomain Component) && (index > 0) [Error]: SecurityErrorDomain::kAboveBoundary if ((id == kOrgUnit) || (id == kDomain Component)) and the index value is greater than the current number of components in the specified attribute

⌋*(RS_CRYPTO_02311)*

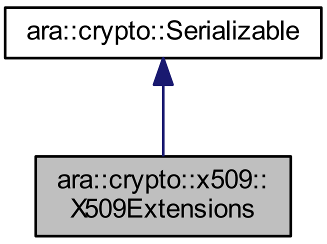
8.22.1.7 class ara::crypto::x509::X509Extensions

[SWS_CRYPT_40500]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::X509Extensions
Scope:	namespace ara::crypto::x509
Base class:	ara::crypto::Serializable
Syntax:	`class X509Extensions : public Serializable {...};`
Header file:	#include "ara/crypto/x509/x509_extensions.h"
Description:	Interface of X.509 Extensions.

⌋*(RS_CRYPTO_02306)*

Inheritance diagram for ara::crypto::x509::X509Extensions:



Public Types

- using Sptr = std::unique_ptr< X509Extensions, CustomDeleter >

Public Member Functions

- virtual std::size_t Count () const noexcept=0

Additional Inherited Members

8.22.1.7.1 Member Typedef Documentation

8.22.1.7.1.1 using ara::crypto::x509::X509Extensions::Sptr = std::unique_ptr<X509Extensions, CustomDeleter>

[SWS_CRYPT_40501]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::X509Extensions::Sptr
Scope:	class ara::crypto::x509::X509Extensions
Derived from:	std::unique_ptr<X509Extensions, CustomDeleter>
Syntax:	`using ara::crypto::x509::X509Extensions::Sptr = std::unique_ptr<X509Extensions, CustomDeleter>;`
Header file:	#include "ara/crypto/x509/x509_extensions.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02404)*

8.22.1.7.2 Member Function Documentation

8.22.1.7.2.1 virtual std::size_t ara::crypto::x509::X509Extensions::Count ( ) const [pure virtual], [noexcept]

[SWS_CRYPT_40511]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Extensions::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_02311)*

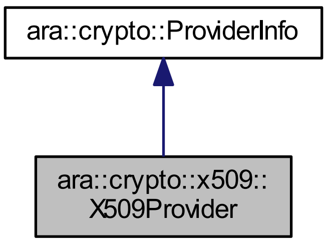
8.22.1.8 class ara::crypto::x509::X509Provider

[SWS_CRYPT_40600]{DRAFT} ⌈

Kind:	class
Symbol:	ara::crypto::x509::X509Provider
Scope:	namespace ara::crypto::x509
Base class:	std::enable_shared_from_this< X509Provider >
Syntax:	`class X509Provider : public enable_shared_from_this< X509Provider > {...};`
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	X.509 Provider interface.
Notes:	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. Any user of this interface should create shared pointers to it only by calls of the method shared_from_this()!

⌋*(RS_CRYPTO_02306)*

Inheritance diagram for ara::crypto::x509::X509Provider:



Public Types

- using Sptr = std::shared_ptr< X509Provider >
- using StorageIndex = std::size_t

Public Member Functions

- virtual ara::core::Result< X509DN::Uptr > CreateEmptyDn (std::size_t capacity) noexcept=0

- virtual ara::core::Result< X509DN::Uptrc > BuildDn (ara::core::StringView dn) noexcept=0
- virtual ara::core::Result< X509DN::Uptrc > DecodeDn (ReadOnlyMemRegion dn, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0
- virtual ara::core::Result< Certificate::Uptr > ParseCert (ReadOnlyMemRegion cert, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0
- virtual ara::core::Result< std::size_t > CountCertsInChain (ReadOnlyMemRegion certChain, Serializable::FormatId formatId=Serializable::kFormatDefault) const noexcept=0
- virtual ara::core::Result< void > ParseCertChain (ara::core::Vector< Certificate::Uptr > &outcome, ReadOnlyMemRegion certChain, Serializable::FormatId formatId=Serializable::kFormatDefault) noexcept=0
- 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
- virtual Certificate::Status VerifyCertByCrl (Certificate &cert) noexcept=0
- virtual Certificate::Status VerifyCertChainByCrl (const ara::core::Vector< Certificate::Uptr > &chain) const noexcept=0
- virtual ara::core::Result< bool > ImportCrl (ReadOnlyMemRegion crl) noexcept=0
- virtual ara::core::Result< void > Import (const Certificate &cert, bool toVolatile=true) noexcept=0
- virtual bool Remove (Certificate::Uptrc &&cert) noexcept=0
- virtual ara::core::Result< void > SaveCertSignRequest (const cryp::X509CertRequest &request, const X509DN &authorityDn, bool toVolatile=true) noexcept=0
- virtual ara::core::Result< void > SetPendingStatus (const CertSignRequest &request) noexcept=0
- virtual ara::core::Result< void > SetAsRootOfTrust (const Certificate &caCert) noexcept=0
- virtual ara::core::Result< OcspRequest::Uptrc > CreateOcspRequest (const Certificate &cert, const cryp::SignerPrivateCtx *signer=nullptr) noexcept=0
- virtual ara::core::Result< OcspRequest::Uptrc > CreateOcspRequest (const ara::core::Vector< const Certificate * > &certList, const cryp::SignerPrivateCtx *signer=nullptr) noexcept=0
- virtual OcspResponse::Uptrc ParseOcspResponse (ReadOnlyMemRegion response) const noexcept=0
- virtual ara::core::Result< bool > CheckCertStatus (Certificate &cert, const OcspResponse &ocspResponse) const noexcept=0

- virtual ara::core::Result< bool > CheckCertStatus (const ara::core::Vector< Certificate ∗ > &certList, const OcspResponse &ocspResponse) const noexcept=0
- virtual Certificate::Uptrc FindCertByDn (const X509DN &subjectDn, const X509DN &issuerDn, time_t validityTimePoint, StorageIndex &certIndex) noexcept=0
- virtual Certificate::Uptrc FindCertByKeyIds (ReadOnlyMemRegion subjectKeyId, ReadOnlyMemRegion authorityKeyId=ReadOnlyMemRegion()) noexcept=0
- virtual Certificate::Uptrc FindCertBySn (ReadOnlyMemRegion sn, const X509DN &issuerDn) noexcept=0
- virtual CertSignRequest::Uptrc FindCertSignRequest (StorageIndex &requestIndex, const X509DN ∗authorityDn=nullptr, const X509DN ∗subjectDn=nullptr, bool pendingCsr=false) noexcept=0
- virtual void CleanupVolatileStorage () noexcept=0

Static Public Attributes

- static const StorageIndex kInvalidIndex = static_cast<std::size_t>(-1LL)

Additional Inherited Members

8.22.1.8.1 Member Typedef Documentation

8.22.1.8.1.1 using ara::crypto::x509::X509Provider::Sptr = std::shared_ptr<X509Provider>

[SWS_CRYPT_40601]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::X509Provider::Sptr
Scope:	class ara::crypto::x509::X509Provider
Derived from:	std::shared_ptr<X509Provider>
Syntax:	`using ara::crypto::x509::X509Provider::Sptr = std::shared_ptr<X509Provider>;`
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	Shared smart pointer of the interface.

⌋*(RS_CRYPTO_02311)*

8.22.1.8.1.2 using ara::crypto::x509::X509Provider::StorageIndex = std::size_t

[SWS_CRYPT_40602]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::crypto::x509::X509Provider::StorageIndex
Scope:	class ara::crypto::x509::X509Provider
Derived from:	std::size_t
Syntax:	`using ara::crypto::x509::X509Provider::StorageIndex = std::size_t;`
Header file:	#include "ara/crypto/x509/x509_provider.h"
Description:	Type of an internal index inside the certificate storage.

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2 Member Function Documentation

8.22.1.8.2.1 virtual ara::core::Result<X509DN::Uptr> ara::crypto::x509::X509Provider::CreateEmptyDn (std::size_t *capacity*) [pure virtual], [noexcept]

[SWS_CRYPT_40611]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::CreateEmptyDn(std::size_t capacity)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<X509DN::Uptr> CreateEmptyDn (std::size_t capacity) noexcept=0;`	
Parameters (in):	capacity	number of bytes that should be reserved for the content of the target X509DN object
Return value:	ara::core::Result< X509DN::Uptr >	Unique smart pointer to created
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.	
Notes:	[Error]: SecurityErrorDomain::kBadAlloc if the requested object can not be allocated dynamically	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.2 virtual ara::core::Result<X509DN::Uptrc> ara::crypto::x509::X509Provider::BuildDn (ara::core::StringView *dn*) [pure virtual], [noexcept]

[SWS_CRYPT_40612]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create completed X.500 Distinguished Name structure from the provided string representation.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if the dn argument has incorrect format [Error]: SecurityErrorDomain::kBadAlloc if the requested object can not be allocated dynamically	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.3 virtual ara::core::Result<X509DN::Uptrc> ara::crypto::x509::X509Provider::DecodeDn (ReadOnlyMemRegion *dn*, Serializable::FormatId *formatId* = Serializable::kFormatDefault) `[pure virtual], [noexcept]`

[SWS_CRYPT_40613]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::DecodeDn(ReadOnlyMemRegion dn, Serializable::FormatId formatId=Serializable::kFormatDefault)	
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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Decode X.500 Distinguished Name structure from the provided serialized format.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if the dn argument cannot be parsed [Error]: SecurityErrorDomain::kUnknownIdentifier if the formatId argument has unknown value [Error]: SecurityErrorDomain::kBadAlloc if the requested object can not be allocated dynamically	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.4 virtual ara::core::Result<Certificate::Uptr> ara::crypto::x509::X509Provider::ParseCert (ReadOnlyMemRegion *cert*, Serializable::FormatId *formatId* = Serializable::kFormatDefault) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40614]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse a serialized representation of the certificate and create its instance.	
Notes:	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! [Error]: SecurityErrorDomain::kInvalidArgument if the cert argument cannot be parsed [Error]: SecurityErrorDomain::kUnknownIdentifier if the formatId argument has unknown value [Error]: SecurityErrorDomain::kBadAlloc if the certificate can not be allocated dynamically	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.5 virtual ara::core::Result<std::size_t> ara::crypto::x509::X509Provider::CountCertsInChain (ReadOnlyMemRegion *certChain*, Serializable::FormatId *formatId* = Serializable::kFormatDefault) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40615]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::x509::X509Provider::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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Count number of certificates in a serialized certificate chain represented by a single BLOB.	
Notes:	[Error]: SecurityErrorDomain::kInvalidArgument if the certChain argument cannot be pre-parsed [Error]: SecurityErrorDomain::kUnknownIdentifier if the formatId argument has unknown value	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.6 virtual ara::core::Result<void> ara::crypto::x509::X509Provider::ParseCertChain (ara::core::Vector< Certificate::Uptr > & *outcome*, ReadOnlyMemRegion *certChain*, Serializable::FormatId *formatId* = Serializable::kFormatDefault) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40616]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::ParseCertChain(ara::core::Vector< Certificate::Uptr > &outcome, 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, 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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse a serialized representation of the certificate chain and create their instances.	

▽

△

Notes:	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). [Error]: SecurityErrorDomain::kInsufficientCapacity if the capacity of outcome vector is less than actual number of certificates in the chain [Error]: SecurityErrorDomain::kInvalidArgument if the certChain argument cannot be parsed [Error]: SecurityErrorDomain::kUnknownIdentifier if the formatId argument has unknown value [Error]: SecurityErrorDomain::kBadAlloc if the certificate can not be allocated dynamically

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.7 virtual ara::core::Result<void> ara::crypto::x509::X509Provider::ParseCertChain (ara::core::Vector< Certificate::Uptr > & *outcome*, const ara::core::Vector< ReadOnlyMemRegion > & *certChain*, Serializable::FormatId *formatId* = Serializable::kFormatDefault) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40617]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse a serialized representation of the certificate chain and create their instances.	

▽

△

Notes:	Off-line validation of the imported 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). [Error]: SecurityErrorDomain::kInsufficientCapacity if capacity of the outcome vector is less than number of elements in the certChain [Error]: SecurityErrorDomain::kInvalidArgument if an element of certChain argument cannot be parsed [Error]: SecurityErrorDomain::kUnknownIdentifier if the formatId argument has unknown value [Error]: SecurityErrorDomain::kBadAlloc if the certificate can not be allocated dynamically

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.8 virtual Certificate::Status ara::crypto::x509::X509Provider::VerifyCertByCrl (Certificate & *cert*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40618]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::VerifyCertByCrl(Certificate &cert)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual Certificate::Status VerifyCertByCrl (Certificate &cert) noexcept=0;`	
Parameters (in):	cert	target certificate for verification
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.	
Notes:	This method updates the Certificate::Status associated with the certificate.	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.9 virtual Certificate::Status ara::crypto::x509::X509Provider::VerifyCertChainByCrl (const ara::core::Vector< Certificate::Uptr > & *chain*) const `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40619]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::VerifyCertChainByCrl(const ara::core::Vector< Certificate::Uptr > &chain)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual Certificate::Status VerifyCertChainByCrl (const ara::core::Vector< Certificate::Uptr > &chain) const noexcept=0;`	
Parameters (in):	chain	target certificate chain for verification
Return value:	Certificate::Status	verification status of the provided certificate chain
Exception Safety:	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.	
Notes:	Verification status of the certificate chain is Certificate::Status::kValid only if all certificates in the chain have such status! Certificates in the chain (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_02311*)

8.22.1.8.2.10 virtual ara::core::Result<bool> ara::crypto::x509::X509Provider::ImportCrl (ReadOnlyMemRegion *crl*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40620]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Import Certificate Revocation List (CRL) or Delta CRL from a file.	
Notes:	[Error]: SecurityErrorDomain::kUnexpectedValue if the provided BLOB is not a CRL/DeltaCRL [Error]: SecurityErrorDomain::kRuntimeFault if the CRL validation has failed	

⌋(*RS_CRYPTO_02311*)

8.22.1.8.2.11 virtual ara::core::Result<void> ara::crypto::x509::X509Provider::Import (const Certificate & *cert*, bool *toVolatile* = `true`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40621]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::Import(const Certificate &cert, bool toVolatile=true)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<void> Import (const Certificate &cert, bool toVolatile=true) noexcept=0;`	
Parameters (in):	cert	a valid certificate that should be imported
	toVolatile	if this flag true then certificate should be saved to the volatile (session) storage, otherwise to the persistent storage
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Import the certificate to volatile or persistent storage.	
Notes:	Only imported certificate may be found by a search and applied for automatic verifications! A certificate can be imported to only one of storages: 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 volatile to the persistent storage 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. [Error]: SecurityErrorDomain::kInvalidArgument if the provided certificate is invalid [Error]: SecurityErrorDomain::kIncompatibleObject if provided certificate has partial collision with a matched CSR in the storage [Error]: SecurityErrorDomain::kContentDuplication if the provided certificate already exists in the storage	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.12 virtual bool ara::crypto::x509::X509Provider::Remove (Certificate::Uptrc && *cert*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40622]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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_02311)*

8.22.1.8.2.13 virtual ara::core::Result<void> ara::crypto::x509::X509Provider::SaveCertSignRequest (const cryp::X509CertRequest & *request*, const X509DN & *authorityDn*, bool *toVolatile* = `true`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40623]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::SaveCertSignRequest(const cryp::X509CertRequest &request, const X509DN &authorityDn, bool toVolatile=true)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<void> SaveCertSignRequest (const cryp::X509CertRequest &request, const X509DN &authorityDn, bool to Volatile=true) noexcept=0;`	
Parameters (in):	request	a valid certificate signature request (that should be send to CA)
	authorityDn	target certification authority (CA) to which the request is addressed
	toVolatile	if this flag true then the request should be saved to the volatile (session) storage, otherwise to the persistent storage
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Save a prepared certificate signing request (CSR) to volatile or persistent storage.	
Notes:	Successfully saved request is located in special area of the storage dedicated for pending requests! Saved CSR obtains status "new" that can be changed to "pending" status when a responsible application send it to correspondent CA (see SetPendingStatus() method). A CSR may be saved only if a trusted certificate from a CA specified by authorityDn exists in the storage. This CA certificate may be trusted as a "root of trust" or via a validated "chain of trust". [Error]: SecurityErrorDomain::kInvalidArgument if the provided certification request is invalid ▽	

▽

△

	△ [Error]: SecurityErrorDomain::kUnknownIdentifier if the provided authority DN is unknown in the system (CA is "registered" in the system only if its certificate is trusted) [Error]: SecurityErrorDomain::kContentDuplication if the provided CSR for same CA already exists in the storage

⌋(*RS_CRYPTO_02311*)

8.22.1.8.2.14 virtual ara::core::Result<void> ara::crypto::x509::X509Provider::SetPendingStatus (const CertSignRequest & *request*) [pure virtual], [noexcept]

[SWS_CRYPT_40624]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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	
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.	
Notes:	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! [Error]: SecurityErrorDomain::kAccessVioalation if the method called by an application without the "CA Connector" permission	

⌋(*RS_CRYPTO_02311*)

8.22.1.8.2.15 virtual ara::core::Result<void> ara::crypto::x509::X509Provider::SetAsRootOfTrust (const Certificate & *caCert*) [pure virtual], [noexcept]

[SWS_CRYPT_40625]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Set specified CA certificate as a "root of trust".	
Notes:	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! [Error]: SecurityErrorDomain::kInvalidArgument if the provided certificate is invalid [Error]: SecurityErrorDomain::kIncompatibleObject if provided certificate is not CA one [Error]: SecurityErrorDomain::kAccessVioalation if the method called by an application without the "Trust Master" permission	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.16 virtual ara::core::Result<OcspRequest::Uptrc> ara::crypto::x509::X509Provider::CreateOcspRequest (const Certificate & *cert*, const cryp::SignerPrivateCtx ∗ *signer* = `nullptr`) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40626]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::CreateOcspRequest(const Certificate &cert, const cryp::SignerPrivateCtx *signer=nullptr)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<OcspRequest::Uptrc> CreateOcspRequest (const Certificate &cert, const cryp::SignerPrivateCtx *signer=nullptr) 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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create OCSP request for specified certificate.	

▽

△

Notes:	This method may be used for implementation of the "OCSP stapling". [Error]: SecurityErrorDomain::kInvalidArgument if the provided certificate is invalid [Error]: SecurityErrorDomain::kIncompleteArgState if the signer context is not initialized by a key

⌋(*RS_CRYPTO_02311*)

8.22.1.8.2.17 virtual ara::core::Result<OcspRequest::Uptrc> ara::crypto::x509::X509Provider::CreateOcspRequest (const ara::core::Vector< const Certificate ∗ > & *certList,* const cryp::SignerPrivateCtx ∗ *signer* = `nullptr`) `[pure virtual], [noexcept]`

[SWS_CRYPT_40627]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::CreateOcspRequest(const ara::core::Vector< const Certificate * > &certList, const cryp::SignerPrivateCtx *signer=nullptr)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<OcspRequest::Uptrc> CreateOcspRequest (const ara::core::Vector< const Certificate * > &certList, const cryp::SignerPrivateCtx *signer=nullptr) 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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Create OCSP request for specified list of certificates.	
Notes:	This method may be used for implementation of the "OCSP stapling". [Error]: SecurityErrorDomain::kInvalidArgument if the provided certificates are invalid [Error]: SecurityErrorDomain::kIncompleteArgState if the signer context is not initialized by a key	

⌋(*RS_CRYPTO_02311*)

8.22.1.8.2.18 virtual OcspResponse::Uptrc ara::crypto::x509::X509Provider::ParseOcspResponse (ReadOnlyMemRegion *response*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40628]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Parse serialized OCSP response and create correspondent interface.	
Notes:	This method may be used for implementation of the "OCSP stapling". [Error]: SecurityErrorDomain::kUnexpectedValue if the provided BLOB response doesn't keep an OCSP response	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.19 virtual ara::core::Result<bool> ara::crypto::x509::X509Provider::CheckCertStatus (Certificate & *cert*, const OcspResponse & *ocspResponse*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40629]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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 certificates list is verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Check certificate status by directly provided OCSP response.	
Notes:	This method may be used for implementation of the "OCSP stapling". This method updates the Certificate::Status associated with the certificate. [Error]: SecurityErrorDomain::kInvalidArgument if the cert is invalid [Error]: SecurityErrorDomain::kRuntimeFault if the ocspResponse is invalid	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.20 virtual ara::core::Result< bool > ara::crypto::x509::X509Provider::CheckCertStatus (const ara::core::Vector< Certificate * > & *certList,* const OcspResponse & *ocspResponse*) const `[pure virtual], [noexcept]`

[SWS_CRYPT_40630]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::CheckCertStatus(const ara::core::Vector< Certificate * > &certList, const OcspResponse &ocspResponse)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual ara::core::Result<bool> CheckCertStatus (const ara::core::Vector< Certificate * > &certList, const OcspResponse &ocspResponse) const noexcept=0;`	
Parameters (in):	certList	a certificates list that should be verified
	ocspResponse	an OCSP response
Return value:	ara::core::Result< bool >	true if the certificates list is verified successfully and false otherwise
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/x509_provider.h"	
Description:	Check status of a certificates list by directly provided OCSP response.	
Notes:	This method may be used for implementation of the "OCSP stapling". This method updates the Certificate::Status associated with the certificates in the list. [Error]: SecurityErrorDomain::kInvalidArgument if the provided certificates are invalid [Error]: SecurityErrorDomain::kRuntimeFault if the ocspResponse is invalid	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.21 virtual Certificate::Uptrc ara::crypto::x509::X509Provider::FindCertByDn (const X509DN & *subjectDn,* const X509DN & *issuerDn,* time_t *validityTimePoint,* StorageIndex & *certIndex*) `[pure virtual], [noexcept]`

[SWS_CRYPT_40631]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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 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 the subject and issuer Distinguished Names (DN).	
Notes:	Argument certIndex represents an internal index of current certificate in the storage. In order to start certificate search from begin: certIndex = kInvalidIndex	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.22 virtual Certificate::Uptrc ara::crypto::x509::X509Provider::FindCertByKeyIds (ReadOnlyMemRegion *subjectKeyId*, ReadOnlyMemRegion *authorityKeyId* = ReadOnlyMemRegion *()*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40632]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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 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 SKID & AKID.	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.23 virtual Certificate::Uptrc ara::crypto::x509::X509Provider::FindCertBySn (ReadOnlyMemRegion *sn*, const X509DN & *issuerDn*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40633]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::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 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_02311)*

8.22.1.8.2.24 virtual CertSignRequest::Uptrc ara::crypto::x509::X509Provider::FindCertSignRequest (StorageIndex & *requestIndex*, const X509DN ∗ *authorityDn* = *`nullptr`*, const X509DN ∗ *subjectDn* = *`nullptr`*, bool *pendingCsr* = *`false`*) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40634]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::X509Provider::FindCertSignRequest(StorageIndex &requestIndex, const X509DN *authorityDn=nullptr, const X509DN *subjectDn=nullptr, bool pendingCsr=false)	
Scope:	class ara::crypto::x509::X509Provider	
Syntax:	`virtual CertSignRequest::Uptrc FindCertSignRequest (StorageIndex &requestIndex, const X509DN *authorityDn=nullptr, const X509DN *subjectDn=nullptr, bool pendingCsr=false) noexcept=0;`	
Parameters (in):	authorityDn	optional authority DN of the target CA that should issue the certificate
	subjectDn	optional subject DN of the request (if only specific subject interested)
	pendingCsr	optional flag that specifies the processing status of the interested requests: "new" (if true) or "pending" (if false)

▽

△

Parameters (inout):	requestIndex	an index of the last found request in the storage, it is dedicated for iteration through all (suitable for the provided filter) CSRs waiting for certificates (input: index of previous found CSR, output: index of current found CSR)
Return value:	CertSignRequest::Uptrc	unique smart pointer to a found certificate signing request 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 signing request (CSR) kept in the storage and waiting for the certificate.	
Notes:	The optional arguments should be used for filtering of the CSRs that will be found. Argument requestIndex represents an internal index of current request in the storage. In order to start requests search from begin: requestIndex = kInvalidIndex	

⌋*(RS_CRYPTO_02311)*

8.22.1.8.2.25 virtual void ara::crypto::x509::X509Provider::CleanupVolatileStorage ( ) `[pure virtual]`, `[noexcept]`

[SWS_CRYPT_40635]{DRAFT} ⌈

Kind:	function
Symbol:	ara::crypto::x509::X509Provider::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.
Notes:	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_02311)*

8.22.1.8.3 Member Data Documentation

8.22.1.8.3.1 const StorageIndex ara::crypto::x509::X509Provider::kInvalidIndex = static_cast<std::size_t>(-1LL) `[static]`

[SWS_CRYPT_40603]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::crypto::x509::X509Provider::kInvalidIndex
Scope:	class ara::crypto::x509::X509Provider
Type:	const StorageIndex
Syntax:	`const StorageIndex ara::crypto::x509::X509Provider::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_02311)*

8.22.2 Enumeration Type Documentation

8.22.2.1 enum ara::crypto::x509::OcspResponseStatus : std::uint8_t `[strong]`

[SWS_CRYPT_40001]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::crypto::x509::OcspResponseStatus	
Scope:	namespace ara::crypto::x509	
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.	
Notes:	Storage type: 8 bit unsigned integer.	

⌋*(RS_CRYPTO_02306)*

8.22.2.2 enum ara::crypto::x509::OcspCertStatus : std::uint8_t `[strong]`

[SWS_CRYPT_40002]{DRAFT} ⌈

Kind:	enumeration
Symbol:	ara::crypto::x509::OcspCertStatus
Scope:	namespace ara::crypto::x509

▽

△

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.	
Notes:	Storage type: 8 bit unsigned integer.	

⌋(*RS_CRYPTO_02306*)

8.22.3 Function Documentation

8.22.3.1 ara::core::Result<X509Provider::Sptr> ara::crypto::x509::LoadX509Provider ( ) [noexcept]

[SWS_CRYPT_40099]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::crypto::x509::LoadX509Provider()	
Scope:	namespace ara::crypto::x509	
Syntax:	`ara::core::Result<X509Provider::Sptr> LoadX509Provider () noexcept;`	
Return value:	ara::core::Result< X509Provider::Sptr >	shared smart pointer to loaded X.509 Provider
Exception Safety:	noexcept	
Thread Safety:	Thread-safe	
Header file:	#include "ara/crypto/x509/entry_point.h"	
Description:	Factory that creates or return existing single instance of the X.509 Provider.	
Notes:	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. [Error]: SecurityErrorDomain::kRuntimeFault if the X.509 Provider cannot be loaded	

⌋(*RS_CRYPTO_02306*)

8.23 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

A Mentioned Class Tables

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.