Document Title	Requirements on Cryptography
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	889

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R21-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2021-11-25	R21-11	AUTOSAR Release Management	• Updated (upward traceability): [RS_CRYPTO_02001] [RS_CRYPTO_02003] [RS_CRYPTO_02003] [RS_CRYPTO_02004] [RS_CRYPTO_02008] [RS_CRYPTO_02009] [RS_CRYPTO_02106] [RS_CRYPTO_02113] • Updated (req. text): [RS_CRYPTO_02209]
2020-11-30	R20-11	AUTOSAR Release Management	• Removed: [RS_CRYPTO_02406] • Updated: [RS_CRYPTO_02201]
2019-11-28	19-11	AUTOSAR Release Management	• Removed: [RS_CRYPTO_02114] [RS_CRYPTO_02311] [RS_CRYPTO_02404] • Updated: [RS_CRYPTO_02009] [RS_CRYPTO_02110]

2019-03-29	19-03	AUTOSAR Release Management	• Editorial changes and rephrasing • Improved requirements description and rationale (Updated : [RS_CRYPTO_02001] [RS_CRYPTO_02002] [RS_CRYPTO_02003] [RS_CRYPTO_02004] [RS_CRYPTO_02005] [RS_CRYPTO_02007] [RS_CRYPTO_02009] [RS_CRYPTO_02109] [RS_CRYPTO_02116] [RS_CRYPTO_02202] [RS_CRYPTO_02206])
2018-10-31	18-10	AUTOSAR Release Management	• Removed : [RS_CRYPTO_02303] and [RS_CRYPTO_02402] • Updated : [RS_CRYPTO_02006]
2018-03-29	18-03	AUTOSAR Release Management	• Existing requirements are corrected • Additional requirements are added
2017-10-27	17-10	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Scope of Document

This document specifies requirements on the Crypto Stack of the AUTOSAR Adaptive Platform.

2 Conventions to be used

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template [1], chapter Support for Traceability.

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template [1], chapter Support for Traceability.

3 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to RS_Cryptography that are not included in the AUTOSAR TR Glossary.

Abbreviation / Acronym:	Description:
HSM	Hardware Software Module
PKI	Public Key Infrastructure
SHE	Secure Hardware Extension
TPM	Trusted Platform Module

Table 3.1: Acronyms and Abbreviations

4 Requirements Specification

4.1 Functional Overview

The AUTOSAR Adaptive Platform provides functionality to perform cryptographic operations by using standardized interfaces and associated modeling.

4.2 Functional Requirements

[RS_CRYPTO_02001]{DRAFT} **The Crypto Stack shall conceal symmetric keys from the users** ⌈

Description:	There shall be no interfaces for the users to directly extract symmetric key values. Keys shall be addressed via identifiers by the users, preventing the key values disclosure.
Rationale:	If symmetric key values are available in the application at runtime it increases the risk of key compromise. If symmetric key values are stored in the application, centralized key management (e.g. renewal) is hard.
Dependencies:	–
Use Case:	Keys are stored in HSMs and never exposed in plain text.
Supporting Material:	–

⌋*(RS_Main_00445, RS_Main_00514, RS_Main_00170)*

[RS_CRYPTO_02002]{DRAFT} **The Crypto Stack shall conceal asymmetric private keys from the users** ⌈

Description:	There shall be no interfaces for the users to directly extract asymmetric private key values. Keys shall be addressed via identifiers by the users, preventing the key values disclosure.
Rationale:	If asymmetric private key values are available in the application at runtime it increases the risk of key compromise. If asymmetric private key values are stored in the application, centralized key management (e.g. renewal) is hard.
Dependencies:	–
Use Case:	Keys are stored in HSMs and never exposed in plain text.
Supporting Material:	–

⌋*(RS_Main_00445, RS_Main_00514, RS_Main_00170)*

[RS_CRYPTO_02003]{DRAFT} **The Crypto Stack shall support management of non-persistent session/ephemeral keys during their lifetime** ⌈

Description:	Some cryptographic keys are only used for a single message or communication session. These keys are referred to as “session keys” (usually for short-term symmetric keys) or “ephemeral keys” (for ephemeral public/private keys in asymmetric key-agreement protocols). The Crypto Stack shall support secure handling of session/ephemeral keys during their lifetime.
Rationale:	The session/ephemeral keys are required for secure implementation of multiple cryptographic protocols. Session/ephemeral keys should not occupy persistent slots due to their transient nature.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02004]{DRAFT} **The Crypto Stack shall support secure storage of cryptographic artifacts** ⌈

Description:	The Crypto Stack shall support secure storage of cryptographic artifacts, including but not limited to the following items: • Secret, Private and Public Keys • Algorithm-specific Domain Parameters • Symmetric or asymmetric Signatures • Password Hashes • Secret Seeds • Certificate Signing Requests • Certificates and Certificate Chains • Certificate Revocation Lists Correspondent protection measures should be applied to each artifact according to its type: confidentiality, integrity, authenticity.
Rationale:	Basic functionality.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00170*)

[RS_CRYPTO_02005]{DRAFT} **The Crypto Stack shall support unique identification of cryptographic objects** ⌈

Description:	The Crypto Stack shall assign and keep a unique identifier to any produced cryptographic artifact that can be saved or exported.
Rationale:	At least the unique identification of cryptographic objects is required for definition of dependencies between different objects. Also the unique identifiers can be used for general searching of concrete instances and prevention of duplication.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410, RS_Main_00514*)

[RS_CRYPTO_02006]{DRAFT} **The Crypto Stack shall support a version control mechanism and distinguish "versions" and "origin sources" of cryptographic objects** ⌈

Description:	The Crypto Stack shall apply a version control mechanism during saving of any cryptographic object. Also it shall provide interfaces for observing version information of any saveable or exportable cryptographic object. At least this information shall include "version number" and "origin source". The information about an object's version should stay actual after provisioning of the object to different ECUs, where it may be kept together with objects obtained from other sources. But a host/ECU that produced an object can ensure uniqueness and sequential order of the "version number" only in its own scope. Therefore additional attribute "origin source" is required and scope of its uniqueness should be global. **Note:** A few logically related objects of different types and generated together (like private and public keys of a single key-pair) must have common version number in order to simplify their versions identification. **Note:** Combination of the global uniqueness of the "origin source" and the local uniqueness of the "version number" (in scope of the source) together means that the version information uniquely identifies the object of specific type. It means that the version information together with the object type uniquely identify each cryptographic object saved in an ECU Key Storage.
Rationale:	The Crypto Stack should prevent the "repetition attacks", when an attacker tries to import/inject again some outdated/compromised and already revoked/substituted object.
Dependencies:	RS_CRYPTO_02005
Use Case:	A key slot owner application may use the version information of an owned object in it's business logic.
Supporting Material:	–

⌋(*RS_Main_00410, RS_Main_00514*)

[RS_CRYPTO_02007]{DRAFT} **The Crypto Stack shall provide means for secure handling of "secret seeds"** ⌈

Description:	The Crypto stack shall provide interfaces for saving, loading, importing and exporting of secret seeds.
Rationale:	The “secret seed” can represent some key material that cannot be directly loaded to a key input of some transformation, but it is used for derivation of concrete “slave” keys. Also the secret seed can be used for loading to a “non-key” input (like salt / nonce / initialization vector) of some cryptographic transformation, but specific application can need to keep it in secret too. For such secret objects the Crypto Stack shall support protection measures similar to the keys. Disclosure of the secret seeds can lead to compromising of whole crypto protocol.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02008]{DRAFT} **The Crypto Stack shall support restrictions of the allowed usage scope for keys and “secret seeds”** ⌈

Description:	The Crypto Stack shall keep the usage restriction information together with correspondent key or secret seed object and use this information every time, when an application tries to load the object to specific transformation context. The allowed usage scope should specify a list of cryptographic transformation types that can be executed using this key or seed object.
Rationale:	The restriction of allowed usage of keys/seeds on the platform level prevents their inappropriate usage by untrusted or compromised applications. In such way, simple “cryptography restriction services” (like “encrypt only”, “decrypt only”, “verify only”, etc.) can be provided without implementation of dedicated services, but just via granting restricted usage access to correspondent keys.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00410*, *RS_Main_00514*, *RS_Main_00170*)

[RS_CRYPTO_02009]{DRAFT} **The Crypto stack shall support separation of applications” access rights for each cryptographic object slot** ⌈

Description:	Adaptive applications should have exclusive access to cryptogaphic object slots. Applications can execute saving and erasing of key slot content. The slot type ”application” allows only the configured application to use the slot contents. If the slot type is ”machine”, the configured application acts only as ”key-manager”, while stack services will be allowed to use the slot content (e.g. for SecOC, TLS).
Rationale:	If two or more applications have the right to update some key slot, then each of them cannot trust to the key slot content, because potentially the content can be updated by a compromised application.
Dependencies:	RS_CRYPTO_02008
Use Case:	Some Key Management application can be in charge of updating “machine” type platform keys.
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00410*, *RS_Main_00514*, *RS_Main_00170*)

[RS_CRYPTO_02101]{DRAFT} **The Crypto Stack shall provide interfaces to generate cryptographic keys for all supported primitives** ⌈

Description:	The Crypto Stack shall support creating cryptographic keys without getting access to the plain key material.
Rationale:	Key confidentiality
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02102]{DRAFT} **The Crypto Stack shall prevent keys from being used in incompatible or insecure ways** ⌈

Description:	The Crypto Stack should detect and prevent use of keys with incompatible algorithms. Keys managed by the Crypto Stack shall be associated with information to detect and prevent use with conflicting or privileged operations.
Dependencies:	–
Use Case:	Protect against unauthorized or incompatible operations that jeapardize confidentiality and integrity of key material (information leakage, key conjuring, API logic attacks).
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02103]{DRAFT} **The Crypto Stack shall support primitives to derive cryptographic key material from a base key material** ⌈

Description:	The Crypto Stack shall support deriving cryptographic keys using a well-defined algorithm from a base key without getting access to the plain key material.
Rationale:	Generating multiple well-defined symmetric keys from a base key
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02104]{DRAFT} **The Crypto Stack shall support a primitive to exchange cryptographic keys with another entity** ⌈

Description:	The Crypto Stack shall support exchanging cryptographic keys without getting access to the plain key material.
Rationale:	Establish common secret
Dependencies:	–
Use Case:	Establish TLS session keys
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02105]{DRAFT} **Symmetric keys and asymmetric private keys shall be imported and exported in a secure format.** ⌈

Description:	The crypto stack shall provide interfaces for import and export of symmetric keys and asymmetric private keys in a secure format.
Rationale:	Support secure distribution of keys from a backend system and/or migration or backup of keys between systems.
Dependencies:	–
Use Case:	Wrapping / unwrapping keys without exposing the key values.
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00150*)

[RS_CRYPTO_02106]{DRAFT} **The Crypto Stack shall provide interfaces for secure processing of passwords** ⌈

Description:	The Crypto Stack shall support password based key derivation and secure password hashing. Passwords should be processed in a manner preventing their disclosure.

▽

△

Rationale:	Passwords are the simplest and widely used method for human users authentication.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00170*)

[RS_CRYPTO_02107]{DRAFT} **The Crypto Stack shall support the algorithm specification in any key generation or derivation request** ⌈

Description:	Interfaces of the Crypto Stack shall support a possibility to provide a full or basic specification of the target cryptographic algorithm for any key generation (symmetric and asymmetric primitives) or key derivation (symmetric primitives only) requests.
Rationale:	Inappropriate usage of a key (including a session key) can lead to leakage of confidential information or other type of compromising.
Dependencies:	RS_CRYPTO_02102
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00410*, *RS_Main_00514*)

[RS_CRYPTO_02108]{DRAFT} **The Crypto Stack shall provide interfaces for management and usage of algorithm-specific domain parameters** ⌈

Description:	Interfaces of the Crypto Stack shall support a possibility to share some common domain parameters for configuration of different primitive's instances. A single set of domain parameters can be used with different key values. In most cases domain parameters are public configuration attribute of an algorithm, but Crypto Stack API should support the confidential storage of domain parameters too.
Rationale:	Most of modern asymmetric cryptographic algorithms use domain parameters, also some symmetric algorithms expects specific configuration parameters. The set of additional parameters required by some algorithm depends from the algorithm only and cannot be predicted in the general primitive's interface.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02109]{DRAFT} **The Crypto Stack shall support interfaces for a unified Machine-wide storage and retrieval of different crypto objects** ⌈

Description:	A wide range of hardware (e.g. HSM/TPM/SHE based) and/or software based (e.g. encrypted files) can be supported for secure storage and retrieval of different crypto objects (e.g. keys, certificates, digests, etc.). Therefore, a unified Machine-wide access to all these different storage providers abstracts physical details about storage handling and reduces complexity of cooperative usage of different crypto objects by applications.
Rationale:	A few trusted applications can have a need to use some keys (or other crypto objects) cooperatively while applications' access rights to the crypto object slots needs to be controlled. A logically centralized crypto object storage handling can facilitate these scenarios conveniently..
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*, *RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02110]{DRAFT} **The Crypto Stack shall support prototyping of application-exclusive key slot resources** ⌈

Description:	The Crypto Stack shall support allocation of key slots during deployment of an application owning correspondent key slots. Access rights and content restrictions of the new key slots should be defined according to the application manifest at the allocation time.
Rationale:	Key slot content restrictions and access rights required by the slots owning application depend on the application design and therefore they should be supplied as a part of application deployment package.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00410*, *RS_Main_00514*)

[RS_CRYPTO_02111]{DRAFT} **The Crypto Stack shall provide applications a possibility to define usage restrictions of any new generated or derived key** ⌈

Description:	Interfaces of the Crypto Stack shall support the possibility to define the allowed usage restrictions of any new generated or derived key.
Rationale:	The usage restrictions of a session key can be defined only by the application itself. Also the key slot prototype can miss or have only partial specification of the content restriction, in such way providing some flexibility to the application.
Dependencies:	RS_CRYPTO_02008
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*, *RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02112]{DRAFT} **The Crypto Stack shall execute export/import of a key value together with its meta information** ⌈

Description:	The Crypto Stack shall execute export/import of a key object together with its whole meta information, which should include: • Unique identifier (at least “origin” and “version”) • Assigned cryptographic algorithm specification • Allowed usage restrictions These information must be part of integrity control of the exported/imported key object and optionally can be encrypted.
Rationale:	The whole key’s meta information is required for its correct application.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*, *RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02113]{DRAFT} **The Crypto Stack interfaces shall support control of the exportability property of a key object** ⌈

Description:	Owner application executing generation or importing of a cryptographic object shall have possibility to restrict the exportability property of the generated/imported object.
Rationale:	Unauthorized export of a key (even in encrypted form) can compromise the system.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00170*)

[RS_CRYPTO_02115]{DRAFT} **The Crypto Stack shall enforce assigning required domain parameters to a key in its generation or derivation procedure** ⌈

Description:	If some cryptographic algorithm requires specification of domain parameters then key generation or key derivation procedures producing key for this algorithm shall enforce direct specification of the domain parameters for the target key. Changing of the domain parameters assigned to an existing key should be impossible. The Crypto Stack implementation may provide some well-known domain parameters specified in some standards via their standardized names.

▽

△

Rationale:	For some asymmetric algorithms specification of a key is possible only in context of concrete domain parameters. Usage of a single (symmetric or asymmetric) key together with different domain parameters of its algorithm can lead to security risks.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*, *RS_Main_00514*)

[RS_CRYPTO_02116]{DRAFT} **The Crypto Stack shall support version control of key objects kept in the Key Storage** ⌈

Description:	A key slot shall allow to define a source of keys and switch on the version control mechanism for this key slot content. The Crypto Stack shall allow saving of a new key object into a key slot with enabled version control, only if the key version will be increased and the source is matching. The version control mechanism must keep the version of the last key saved in the slot even after erasing of the key value.
Rationale:	The basic version control logic must be implemented by the Crypto Stack to enable rollback protection in a transparent way for applications.
Dependencies:	RS_CRYPTO_02109, RS_CRYPTO_02110
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00150*, *RS_Main_00514*)

[RS_CRYPTO_02201]{DRAFT} **The Crypto Stack shall provide interfaces to use symmetric encryption and decryption primitives** ⌈

Description:	The Crypto Stack shall support encrypting and decrypting data using an algorithm for symmetric encryption/decryption primitives.
Rationale:	Encrypted data
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02202]{DRAFT} **The Crypto Stack shall provide interfaces to use asymmetric encryption and decryption primitives** ⌈

Description:	The Crypto Stack shall support encrypting and decrypting data using an asymmetric algorithm.
Rationale:	While encryption/decryption of bulk data (long messages) should be done using symmetric-key algorithms for efficiency reasons, the Crypto Stack supports also asymmetric encryption/decryption primitives required by special use cases that apply asymmetric encryption/deception on messages of short length and to facilitate implementing standards that include hybrid encryption/decryption schemes.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02203]{DRAFT} **The Crypto Stack shall provide interfaces to use message authentication code primitives** ⌈

Description:	The Crypto Stack shall support creating and verifying message authentication codes (MAC).
Rationale:	SecOC using MACs to authenticate messages
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02204]{DRAFT} **The Crypto Stack shall provide interfaces to use digital signature primitives** ⌈

Description:	The Crypto Stack shall support creating and verifying digital signatures.
Rationale:	Digitally signed updates
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02205]{DRAFT} **The Crypto Stack shall provide interfaces to use hashing primitives** ⌈

Description:	The Crypto Stack shall support creating and verifying cryptographic hashes.
Rationale:	Signature verification
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02206]{DRAFT} **The Crypto Stack shall provide interfaces to configure and use random number generation** ⌈

Description:	The Crypto Stack shall support generating cryptographically strong random numbers.
Rationale:	Random numbers are required to generate cryptographic keys, nonces and other inputs to cryptographic protocols.
Dependencies:	–
Use Case:	Once configured, random number generator is used by different primitives.
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02207]{DRAFT} **The Crypto Stack shall provide interfaces to use authenticated symmetric encryption and decryption primitives** ⌈

Description:	The Crypto Stack shall support encrypting and decrypting data using an algorithm for authenticated symmetric encryption/decryption primitives.
Rationale:	Authenticated encrypted data
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02208]{DRAFT} **The Crypto Stack shall provide interfaces to use symmetric key wrapping primitives** ⌈

Description:	The Crypto Stack shall support symmetric authenticated encrypting/decrypting or wrapping/unwrapping of key values unavailable for applications in a plain form.
Rationale:	Secure keys transportation.
Dependencies:	RS_CRYPTO_02001, RS_CRYPTO_02002

▽

△

Use Case:	Export/Import of key material.
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02209]{DRAFT} **The Crypto Stack shall provide interfaces to use asymmetric key encapsulation primitives** ⌈

Description:	The Crypto Stack shall support asymmetric key encapsulation mechanism for secure transportation of key values.
Rationale:	Secure keys transportation.
Dependencies:	RS_CRYPTO_02001, RS_CRYPTO_02002, RS_CRYPTO_02208
Use Case:	Export/Import of key material.
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00514*, *RS_Main_00410*)

[RS_CRYPTO_02301]{DRAFT} **The Crypto Stack API shall provide a standardized header files structure** ⌈

Description:	The application shall use standardized header files to abstract from the underlying implementation and platform.
Rationale:	The applications code shall be reusable across different implementations of the AUTOSAR Adaptive platform.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00060*)

[RS_CRYPTO_02302]{DRAFT} **The Crypto Stack API shall support a streaming approach** ⌈

Description:	Some primitives are generally used to process large amounts of data. This data may be streamed into the Crypto Stack in multiple smaller pieces.
Rationale:	Basic functionality
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*)

[RS_CRYPTO_02304]{DRAFT} **The Crypto Stack API should support the possibility to move a state of a "counter mode" stream cipher to a random position** ⌈

Description:	The Crypto Stack API should support the possibility to utilize the especial benefit of stream ciphers in the "counter mode" (like CTR or GCM) to move their states to random positions directly.
Rationale:	Basic functionality, e.g. it is required for "on-the-fly" encryption/decryption of a large data storage.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00410)*

[RS_CRYPTO_02305]{DRAFT} **The Crypto Stack design shall separate cryptographic API from key access API** ⌈

Description:	The Crypto Stack interfaces providing cryptographic transformations should be logically separated from interfaces providing access control to key slots of the permanent Key Storage.
Rationale:	The key access functionality supposes interaction with the IAM framework, but the cryptography implementation independent from this. Therefore separation of these two functional sub-domains simplifies implementation, support and extending of the whole Crypto Stack. Each of these sub-domains can be upgraded independently from another one.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00410, RS_Main_00514)*

[RS_CRYPTO_02306]{DRAFT} **The Crypto Stack shall support integration with a Public Key Infrastructure (PKI)** ⌈

Description:	The Crypto Stack shall support integration with a Public Key Infrastructure (PKI). For this reason it shall provide interfaces for at least: certificate parsing and verification, validation of certificate chains, creation of Certificate Signing Requests (CSR), storing and updating Certificate Revocation Lists (CRL) and Delta CRLs for following usage by the stack, certificate validation via the Online Certificate Status Protocol (OCSP), ordering and transmission of certificates in certificate chains (full or partial), updating a defined set of root certificates.
Rationale:	PKI is a widely used modern mean to facilitate the secure electronic transfer of information between untrusted parties for a range of network activities.
Dependencies:	–

▽

△

Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00060*, *RS_Main_00514*)

[RS_CRYPTO_02307]{DRAFT} **The Crypto Stack design shall separate cryptographic API from the PKI API** ⌈

Description:	The Crypto Stack interfaces providing cryptographic transformations should be logically separated from interfaces providing PKI related functionality.
Rationale:	Main responsibility of the PKI functional domain is parsing and production of data structures in specific formats. Functionally, the PKI is a “consumer” of a cryptography implementation, and main functionality of the client-side PKI uses key-less or public key cryptographic transformations, i.e. it doesn’t need utilization of isolated private/secret contexts. Each of these sub-domains can be upgraded independently from another one.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*, *RS_Main_00514*)

[RS_CRYPTO_02308]{DRAFT} **The Crypto Stack shall support a unified cryptographic primitives naming convention, common for all suppliers** ⌈

Description:	The Crypto Stack should provide interfaces for mapping of unified (Crypto Stack supplier independent) cryptographic primitives’ names to some supplier specific ones.
Rationale:	Introduction of the unified naming convention allows to enable development of portable application source code.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00060*, *RS_Main_00150*, *RS_Main_00410*)

[RS_CRYPTO_02309]{DRAFT} **The Crypto Stack API shall support the run-time configurable usage style** ⌈

Description:	A consumer application should have a possibility to select concrete cryptographic primitives and find out all their properties at run-time.

▽

△

Rationale:	In some use cases an application may not know in advance which concrete primitive it will use for data processing. For example this information can stay available after some “handshake” protocol execution only. Also the possibility to observe properties of currently used object or context is very useful for the application debugging.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*)

[RS_CRYPTO_02401]{DRAFT} **The Crypto Stack should support a joint usage of multiple back-end cryptography providers including ones with non-extractable keys** ⌈

Description:	The Crypto Stack interfaces should support simultaneous cooperative usage of multiple software or hardware based cryptography implementations, which can implement the concept of non-extractable keys (HSMs/TPMs).
Rationale:	Single ECU can have a few different HSMs/TPMs and additional software implementation of cryptography for usage in different application domains.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*, *RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02403]{DRAFT} **The Crypto Stack shall support isolating keys and requests** ⌈

Description:	In a multi-tenant scenario the Crypto Stack shall implement an individual logical view of available session keys and active operations for each tenant.
Rationale:	A application using the Crypto Stack should not be able to observe or manipulate the list of active keys and crypto operations of another application (error injection, timing side-channels, etc.).
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00514*, *RS_Main_00445*)

[RS_CRYPTO_02405]{DRAFT} **The Crypto Stack shall support the key slots identification in a way independent from a concrete deployment** ⌈

Description:	The Crypto Stack shall support some type of unique logical key slot identifiers definable by application designers/developers.
Rationale:	Application needs some simple identification mechanism of logical key slots that is independent from the deployment results, so that these slots identifiers can be directly defined in the executable code.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00060*, *RS_Main_00150*, *RS_Main_00410*)

4.3 Non-Functional Requirements

[RS_CRYPTO_02310]{DRAFT} **The Crypto Stack API shall support an efficient mechanism of error states notification** ⌈

Description:	The Crypto Stack should deliver comprehensive information about an error state what was detected. This information should be enough to recognize the error conditions and make decision how to recover from the error state and continue execution. The delivering mechanism should be convenient for applications' developers and satisfy the Autosar AP C++14 Coding Guidelines. Note: The error states are not expected to be seen in normal program execution.
Rationale:	Basic functionality.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00060*)

5 Requirements Tracing

The following table references the features specified in [2] and links to the fulfillments of these.

Feature	Description	Satisfied by
[RS_Main_00060]	AUTOSAR shall provide a standardized software interface for communication between Applications	[RS_CRYPTO_02301] [RS_CRYPTO_02306] [RS_CRYPTO_02308] [RS_CRYPTO_02310] [RS_CRYPTO_02405]
[RS_Main_00150]	AUTOSAR shall support the deployment and reallocation of AUTOSAR Application Software	[RS_CRYPTO_02105] [RS_CRYPTO_02116] [RS_CRYPTO_02308] [RS_CRYPTO_02405]
[RS_Main_00170]	AUTOSAR shall provide secure access to ECU data and services	[RS_CRYPTO_02001] [RS_CRYPTO_02002] [RS_CRYPTO_02004] [RS_CRYPTO_02008] [RS_CRYPTO_02009] [RS_CRYPTO_02106] [RS_CRYPTO_02113]
[RS_Main_00410]	AUTOSAR shall provide specifications for routines commonly used by Application Software to support sharing and optimization	[RS_CRYPTO_02005] [RS_CRYPTO_02006] [RS_CRYPTO_02008] [RS_CRYPTO_02009] [RS_CRYPTO_02107] [RS_CRYPTO_02109] [RS_CRYPTO_02110] [RS_CRYPTO_02111] [RS_CRYPTO_02112] [RS_CRYPTO_02115] [RS_CRYPTO_02201] [RS_CRYPTO_02202] [RS_CRYPTO_02203] [RS_CRYPTO_02204] [RS_CRYPTO_02205] [RS_CRYPTO_02206] [RS_CRYPTO_02207] [RS_CRYPTO_02208] [RS_CRYPTO_02209] [RS_CRYPTO_02302] [RS_CRYPTO_02304] [RS_CRYPTO_02305] [RS_CRYPTO_02307] [RS_CRYPTO_02308] [RS_CRYPTO_02309] [RS_CRYPTO_02401] [RS_CRYPTO_02405]

[RS_Main_00445]	AUTOSAR shall standardize access to crypto-specific HW and SW	[RS_CRYPTO_02001] [RS_CRYPTO_02002] [RS_CRYPTO_02003] [RS_CRYPTO_02004] [RS_CRYPTO_02007] [RS_CRYPTO_02008] [RS_CRYPTO_02009] [RS_CRYPTO_02101] [RS_CRYPTO_02102] [RS_CRYPTO_02103] [RS_CRYPTO_02104] [RS_CRYPTO_02105] [RS_CRYPTO_02106] [RS_CRYPTO_02107] [RS_CRYPTO_02108] [RS_CRYPTO_02109] [RS_CRYPTO_02110] [RS_CRYPTO_02111] [RS_CRYPTO_02112] [RS_CRYPTO_02113] [RS_CRYPTO_02201] [RS_CRYPTO_02202] [RS_CRYPTO_02203] [RS_CRYPTO_02204] [RS_CRYPTO_02205] [RS_CRYPTO_02206] [RS_CRYPTO_02207] [RS_CRYPTO_02208] [RS_CRYPTO_02209] [RS_CRYPTO_02401] [RS_CRYPTO_02403]
[RS_Main_00514]	AUTOSAR shall support the development of secure systems	[RS_CRYPTO_02001] [RS_CRYPTO_02002] [RS_CRYPTO_02003] [RS_CRYPTO_02004] [RS_CRYPTO_02005] [RS_CRYPTO_02006] [RS_CRYPTO_02007] [RS_CRYPTO_02008] [RS_CRYPTO_02009] [RS_CRYPTO_02101] [RS_CRYPTO_02102] [RS_CRYPTO_02103] [RS_CRYPTO_02104] [RS_CRYPTO_02105] [RS_CRYPTO_02106] [RS_CRYPTO_02107] [RS_CRYPTO_02108] [RS_CRYPTO_02109] [RS_CRYPTO_02110] [RS_CRYPTO_02111] [RS_CRYPTO_02112] [RS_CRYPTO_02113] [RS_CRYPTO_02115] [RS_CRYPTO_02116]

		[RS_CRYPTO_02201] [RS_CRYPTO_02202] [RS_CRYPTO_02203] [RS_CRYPTO_02204] [RS_CRYPTO_02205] [RS_CRYPTO_02206] [RS_CRYPTO_02207] [RS_CRYPTO_02208] [RS_CRYPTO_02209] [RS_CRYPTO_02305] [RS_CRYPTO_02306] [RS_CRYPTO_02307] [RS_CRYPTO_02401] [RS_CRYPTO_02403]

6 References

[1] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[2] Requirements on AUTOSAR Features
AUTOSAR_RS_Features