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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-03

Document Change History			
Date	**Release**	**Changed by**	**Description**
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 Requirements Specification

3.1 Functional Overview

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

3.2 Crypto Stack

[RS_CRYPTO_02001] The Crypto Stack shall conceal symmetric keys from the users ⌈

Type:	draft
Description:	There shall be no interfaces for the users to work with symmetric key values directly. The symmetric key values shall be hidden, but usable via “opaque” interfaces, preventing the key values disclosure. Symmetric key values shall only be exported in a secure format.
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:	–
Supporting Material:	–

⌋(*RS_Main_00330*)

[RS_CRYPTO_02002] The Crypto Stack shall conceal asymmetric private keys from the users ⌈

Type:	draft
Description:	There shall be no interfaces for the users to work with asymmetric private key values directly. The asymmetric private key values shall be hidden, but usable via “opaque” interfaces, preventing the key values disclosure. Asymmetric private key values shall only be exported in a secure format.
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:	–
Supporting Material:	–

⌋(*RS_Main_00330*)

[RS_CRYPTO_02003] The Crypto Stack shall support generation and usage of session keys ⌈

Type:	draft

Description:	Interfaces of the Crypto Stack should support generation of session symmetric and asymmetric keys. Usage of session and permanent keys should be similar for applications. The Crypto Stack shall ensure the secure destruction of the session keys after finishing of their usage.
Rationale:	• The session or “one-time” keys are required for implementation of multiple cryptographic protocols. • Session keys should not occupy persistent slots due to their transient nature.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00200*, *RS_Main_00330*, *RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02004] The Crypto Stack shall support secure storage of different cryptographic artifacts ⌈

Type:	draft
Description:	The Crypto Stack shall support secure storage of the following cryptographic artifacts: • 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_00330*, *RS_Main_00514*)

[RS_CRYPTO_02005] The Crypto Stack shall support unique identification of cryptographic objects ⌈

Type:	draft

Description:	The Crypto Stack shall assign and keep a globally unique identifier to any produced cryptographic artifact, if it can be saved or exported (i.e. if it may be referenced in the future).
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] The key objects' identifiers supported by Crypto Stack shall provide information about "origin" and "version" of the key ⌈

Type:	draft
Description:	The Crypto Stack shall provide interfaces for observing a version and a source of any saveable or exportable key. The uniqueness of the identifiers should stay actual after provisioning of a key to different ECUs. Therefore uniqueness of the "origin" should be global. Similar approach about unique identification of "origin" should be applied to other cryptographic artifacts too.
Rationale:	An application logic can depend on from the current version of a key and/or its source. At least, a key slot owner application requires it. Also it is impossible to talk about the version of a key without knowledge about its origin.
Dependencies:	RS_CRYPTO_02005
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*, *RS_Main_00514*)

[RS_CRYPTO_02007] The Crypto Stack shall support handling of "secret seeds" in a way similar to secret keys ⌈

Type:	draft
Description:	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.
Rationale:	Disclosure of the secret seeds can lead to compromising of whole crypto protocol.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00330*, *RS_Main_00514*)

[RS_CRYPTO_02008] The Crypto Stack shall support restrictions of the allowed usage scope for keys and "secret seeds" ⌈

Type:	draft
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_00330, RS_Main_00410, RS_Main_00514)*

[RS_CRYPTO_02009] For each key slot the Crypto Stack shall support separation of applications access rights on two categories: owner and users ⌈

Type:	draft
Description:	Only single "owner" application should be responsible for maintenance of consistency of the key slot content. The "owner" application can execute saving and erasing of key slot content. All other applications can be granted by the "user" access right only. The "user" access allow loading of the key slot content to a cryptographic transformation context (according to defined key usage permissions). The "owner" application also can have some "user" permissions. The list of "user" applications should be coordinated with the "owner" application supplier, because if just "user" access is granted to an application untrusted by the "owner", then for the "owner" application it means compromising.
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. Therefore support of such use-case requires application of a non-trivial synchronization logic and integrity / authentication control. Adding an untrusted application to the "users" list can compromise all other applications. Allowing the "user" access to a key slot to third-party application is a simplest way to provide a "cryptographic service" without dedicated service implementation (it is applicable in simplest use-cases only), but inappropriate granting access to private or secret keys can lead to the application or whole system compromising. Therefore the implementation of a dedicated service should be a preferable way to provide a cryptographic service to third-party applications.
Dependencies:	RS_CRYPTO_02008
Use Case:	Some Key Management application can be "Owner" of all platform specific keys.
Supporting Material:	–

⌋(*RS_Main_00330*, *RS_Main_00410*, *RS_Main_00514*)

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

Type:	draft
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_00330*, *RS_Main_00514*)

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

Type:	draft
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_00514*, *RS_Main_00330*)

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

Type:	draft
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_00330*, *RS_Main_00514*)

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

Type:	draft
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_00330*, *RS_Main_00514*)

[RS_CRYPTO_02105] The Crypto Stack shall provide interfaces to import and export cryptographic keys into or from a local key storage or a session objects ⌈

Type:	draft
Description:	The Crypto Stack shall support importing and exporting cryptographic keys without getting access to the plain key material. The export and import procedures should ensure confidentiality, integrity and authenticity of the exported/imported key object. The Crypto Stack shall support exporting key objects of both life time categories: persistent or session. The Crypto Stack shall support importing of key objects to persistent or session ones (according to an application preference).
Rationale:	Support secure distribution of keys from a backend system and/or migration or backup of keys between systems.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00445*, *RS_Main_00330*, *RS_Main_00514*, *RS_Main_00150*)

[RS_CRYPTO_02106] The Crypto Stack shall provide interfaces for secure processing of passwords ⌈

Type:	draft
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_00330*, *RS_Main_00514*)

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

Type:	draft
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_00330*, *RS_Main_00410*, *RS_Main_00514*)

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

Type:	draft
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] The Crypto Stack shall support a logically centralized "Key Storage" for all cryptographic artifacts used on the ECU ⌈

Type:	draft
Description:	An application should maximally abstract from details where and how a concrete key (or other cryptographic artifacts) is stored physically, but the Crypto Stack should support cooperative usage of any artifacts by applications. Physically the "Key Storage" can be presented by a few storage instances with different levels of protection, e.g. HSM/TPM internal storage can be used simultaneously with a key storage hosted in encrypted files.
Rationale:	A few trusted applications can have a need to use some keys (or other artifacts) cooperatively. Applications' access rights to all key slots should be controlled by the IAM. Therefore the logically centralized "Key Storage" is more convenient in implementation and usage.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00330*, *RS_Main_00410*, *RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02110] The Crypto Stack shall support the prototyping of applications' access rights and content restrictions of key slot resources ⌈

Type:	draft

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_00330*, *RS_Main_00410*, *RS_Main_00514*)

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

Type:	draft
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_00330*, *RS_Main_00410*, *RS_Main_00514*)

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

Type:	draft
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_00330*, *RS_Main_00410*, *RS_Main_00514*)

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

Type:	draft
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_00330*, *RS_Main_00514*)

[RS_CRYPTO_02114] The Crypto Stack shall support the possibility to restrict the allowed usage of a key individually to each user application ⌈

Type:	draft
Description:	The Crypto Stack shall support individual restrictions of the allowed key usage for each user application.
Rationale:	This approach extends the access rights configuration flexibility.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00330*, *RS_Main_00410*, *RS_Main_00514*)

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

Type:	draft
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] The Crypto Stack shall support the version control of key objects kept in the Key Storage according to the key slot prototype ⌈

Type:	draft

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

⌋(*RS_Main_00150*, *RS_Main_00514*)

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

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

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

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

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

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

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

Type:	draft
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] The Crypto Stack shall provide interfaces to use digital signature primitives ⌈

Type:	draft
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] The Crypto Stack shall provide interfaces to use hashing primitives ⌈

Type:	draft
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] The Crypto Stack shall provide interfaces to configure and use random number generation primitives directly or indirectly by other primitives ⌈

Type:	draft
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] The Crypto Stack shall provide interfaces to use authenticated symmetric encryption and decryption primitives ⌈

Type:	draft
Description:	The Crypto Stack shall support encrypting and decrypting data using an algorithm for authenticated symmetric encryption/decryption primitives.
Rationale:	Authenticated encrypted communication

Dependencies:	–
Use Case:	–
Supporting Material:	–

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

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

Type:	draft
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] The Crypto Stack shall provide interfaces to use asymmetric key encapsulation primitives ⌈

Type:	draft
Description:	The Crypto Stack shall support asymmetric key encapsulation mechanism for secure transportation of key values. It should be possible to use this mechanism together with the symmetric key wrapping.
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] The Crypto Stack API shall provide a standardized header files structure ⌈

Type:	draft
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] The Crypto Stack API shall support a streaming approach ⌈

Type:	draft
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_02303] The Crypto Stack API shall support asynchronous operation ⌈

Type:	draft
Description:	Some primitives may take a long time to complete and thus completion may be communicated in an asynchronous fashion.
Rationale:	Basic functionality
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00410*)

[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 ⌈

Type:	draft
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] The Crypto Stack design shall separate cryptographic API from key access API ⌈

Type:	draft
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] The Crypto Stack shall support integration with a Public Key Infrastructure (PKI) ⌈

Type:	draft
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] The Crypto Stack design shall separate cryptographic API from the PKI API ⌈

Type:	draft
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] The Crypto Stack shall support a unified cryptographic primitives naming convention, common for all suppliers ⌈

Type:	draft
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] The Crypto Stack API shall support the run-time configurable usage style ⌈

Type:	draft
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_02310] The Crypto Stack API shall support an efficient mechanism of error states notification ⌈

Type:	draft
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*)

[RS_CRYPTO_02311] The Crypto Stack API specification should be complete and allow flexible usage of the stack functionality ⌈

Type:	draft

Description:	The Crypto Stack API should have complete specification, i.e. all referenced interfaces and generic data types should be specified explicitly. A set of interfaces, methods and types specified by the Crypto Stack API shall allow efficient usage of the Crypto Stack functionality in different tasks (use cases).
Rationale:	Basic functionality.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00060*, *RS_Main_00200*)

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

Type:	draft
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_00330*, *RS_Main_00410*, *RS_Main_00445*, *RS_Main_00514*)

[RS_CRYPTO_02402] The Crypto Stack shall support prioritizing the processing of requests ⌈

Type:	draft
Description:	Accessing specific functionality can lead to contention that shall be resolved by assigning priorities to requests.
Rationale:	Basic functionality
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00200*)

[RS_CRYPTO_02403] The Crypto Stack shall support isolating keys and requests ⌈

Type:	draft
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_00330*)

[RS_CRYPTO_02404] Design of the Crypto Stack interfaces shall support minimization of resources consumption ⌈

Type:	draft
Description:	Design of the Crypto Stack interfaces shall support consumptions minimization of storage and computational resources: • It should be possible to reuse out-of-use memory regions or hardware computational units in following computing. • Granularity of transformations should allow re-usage of already computed artifacts in following transformations. • Interfaces design should allow an optimal usage of transformations, i.e. without any extra/useless calculations. • Interfaces design should support minimization of required copy operations. • Interfaces design should define supplementary data structures of minimal sizes.
Rationale:	Efficient implementation.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00200*)

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

Type:	draft
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*)

[RS_CRYPTO_02406] The Crypto Stack shall support its efficient usage in the "real-time" and "non-real-time" modes ⌈

Type:	draft

Description:	The Crypto Stack implementation shall enable its efficient usage in the “real-time” mode (when any resources allocation request should be processed in deterministic time) and in the “non-real-time” mode (when processing of the resources allocation requests is not limited by the deterministic time).
Rationale:	The Crypto Stack can be used by applications working in any of mentioned modes (“real-time” or “non-real-time”).
Dependencies:	–
Use Case:	–
Supporting Material:	–

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

4 Requirements Tracing

The following table references the features specified in [**?**,] 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_02311] [RS_CRYPTO_02405] [RS_CRYPTO_02406]
[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_CRYPTO_02406]
[RS_Main_00200]	AUTOSAR specifications shall allow resource efficient implementations	[RS_CRYPTO_02003] [RS_CRYPTO_02311] [RS_CRYPTO_02402] [RS_CRYPTO_02404]
[RS_Main_00330]	No description	[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_02109] [RS_CRYPTO_02110] [RS_CRYPTO_02111] [RS_CRYPTO_02112] [RS_CRYPTO_02113] [RS_CRYPTO_02114] [RS_CRYPTO_02401] [RS_CRYPTO_02403]

[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_02114] [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_02303] [RS_CRYPTO_02304] [RS_CRYPTO_02305] [RS_CRYPTO_02307] [RS_CRYPTO_02308] [RS_CRYPTO_02309] [RS_CRYPTO_02401] [RS_CRYPTO_02405] [RS_CRYPTO_02406]
[RS_Main_00445]	AUTOSAR shall standardize access to crypto-specific HW and SW	[RS_CRYPTO_02003] [RS_CRYPTO_02101] [RS_CRYPTO_02103] [RS_CRYPTO_02104] [RS_CRYPTO_02105] [RS_CRYPTO_02108] [RS_CRYPTO_02109] [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_Main_00514]	AUTOSAR shall support the development of secure systems	[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_02114] [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]

5 References

[1] System Template
AUTOSAR_TPS_SystemTemplate