**Crypto**

# **Introduction**

Cryptography is the practice and study of hiding information. It is the Art or Science of converting a plain intelligible data into an unintelligible data and again retransforming that message into its original form. It provides Confidentiality, Integrity, and Accuracy.

The FC Crypto is Adaptive Platform module offers applications and other Adaptive AUTOSAR Functional Cluster a standardized interface, which provides operations for cryptographic and related calculations.

These operations include cryptographic operations, key management, and certificate handling. FC Crypto manages the actual implementations of all operations, the configuration, and the brokering of operations from applications to implementations. The standardized interface is exposed by the CryptoAPI.

The FC Crypto and its CryptoAPI supports both public-key and symmetric-key cryptography. It allows applications to use mechanisms such as authentication, encryption, and decryption for automotive services.

# **Functional Specification**

The AUTOSAR Adaptive architecture organizes the software of the AUTOSAR Adaptive foundation as functional clusters. These clusters offer common functionality as services to the applications. The Functional Cluster Cryptography (FC Crypto) is part of the AUTOSAR Adaptive Platform Foundation.

The FC Crypto provides the infrastructure to access multiple implementations of cryptographic operations through a standardized interface, CryptoAPI. Operations provided by FC Crypto are grouped into different providers, each of them implements specific domain of cryptography-related functionality:

* CryptoProvider
* KeyStorageProvider
* X.509 Certificate Management Provider

# **Architecture Concepts**

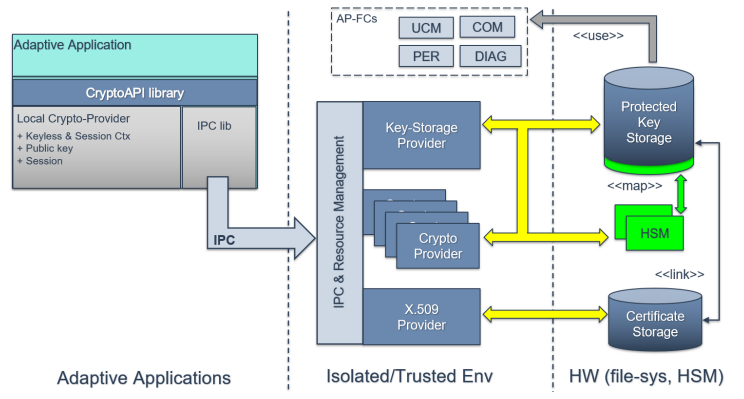
The interfaces defined by FC Crypto are designed to enable integration of 3rd party cryptographic libraries and hardware-based elements. This facilitates implementation of a security trust anchor or acceleration of cryptographic transformations in situations, where the FC Crypto’s default crypto-library will not provide the necessary primitives or hardware acceleration is needed.

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

* **Authentication**: FC Crypto provides mechanisms that allow adaptive applications or functional clusters to prove their identity to other applications or functional clusters.
* **Non-Repudiation**: FC Crypto supports the concept of non-repudiation, where someone cannot deny the validity of something.
* **Confidentiality**: FC Crypto allows to keep information private. Cryptographic systems were originally developed to function in this capacity. Whether it be system or user specific data sent during system debugging or tracing, or storing confidential vehicle / ECU data, encryption can assure that only users who have access to the appropriate key will get read access to the data plaintext.
* **Integrity**: FC Crypto ensures that secured data is not altered during storage or transmission without the receiver detecting this altering. Additionally, FC Crypto allows applications to build functionality, which guarantees the integrity of elements or services.

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

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

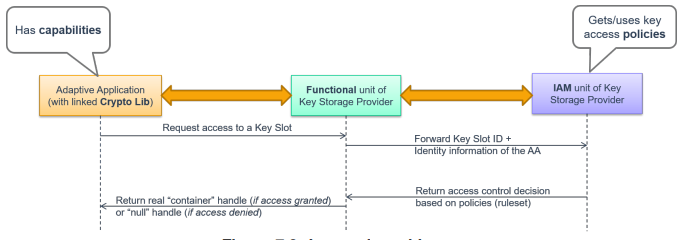


The architectural overview shows all parts, such as X.509 Provider for certificate handling, Crypto Provider and Key Storage Provider. Above depicts the high-level architecture of FC Crypto including the previously described elements.

## **Integration with Identity and Access Management**

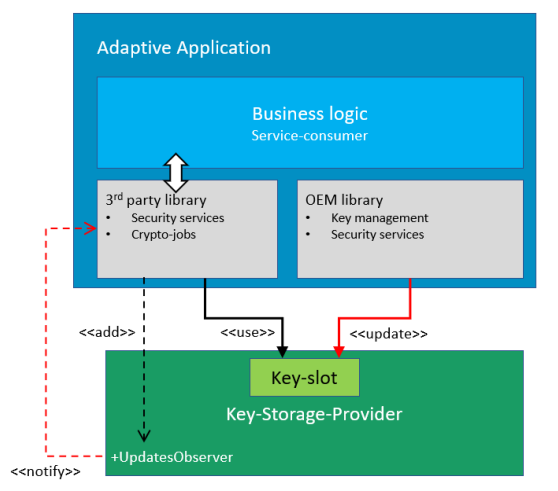
To enable access control the FC Crypto shall implement a Policy Enforcement Point (PEP) to enforce the policy decision obtained from the Policy Decision Point (PDP) as specified by Identity and Access Management (IAM). Thus, an interaction is needed between FC Crypto (PEP) and some entity that implements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

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



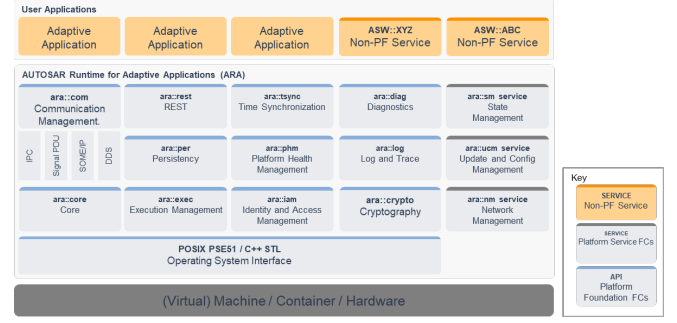
IAM enables access control to modelled entities or resources. Currently, FC Crypto considers access control only for two types of resources: Key Slot (read/write) and Certificate Slot (write). To simplify IAM configuration FC Crypto specifies the exclusive-access-model, which states that access to a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

Below figure showcases the scenario : The Adaptive Application has exclusive-access to a Key Slot, which is used by a library providing cryptographic services to a higher layer (business logic). At the same time another library independently manages Key Slot content (e.g. crypto-keys).



## **Integration into AUTOSAR**

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



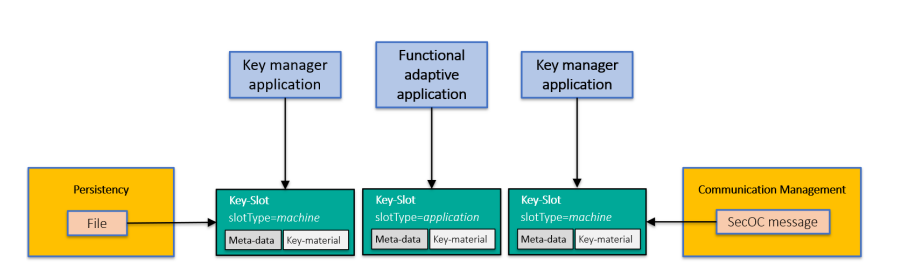
Their differential treatment is due to the underlying trust-model: system services (Functional Clusters) are the trusted foundation while Adaptive Applications are untrusted additions.

To ensure secure access from application side the trust-model, in the form of IAM, is designed for and applied only to Adaptive Applications.

Similarly, the exclusive-access-model aims at protecting an application’s own resources against access by other applications, but additionally also against access by functional clusters other than FC Crypto. On the other hand, some functional clusters specify their own key-slots, which contain key-material to be used when implementing certain system services (e.g. secure data storage, secure diagnostics or secure communication such as SecOC). Because key-management of Key Slots used by functional clusters should be possible from an Adaptive Application (e.g. OEM key manager), the exclusive-access-model defines two types of Key Slots:

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

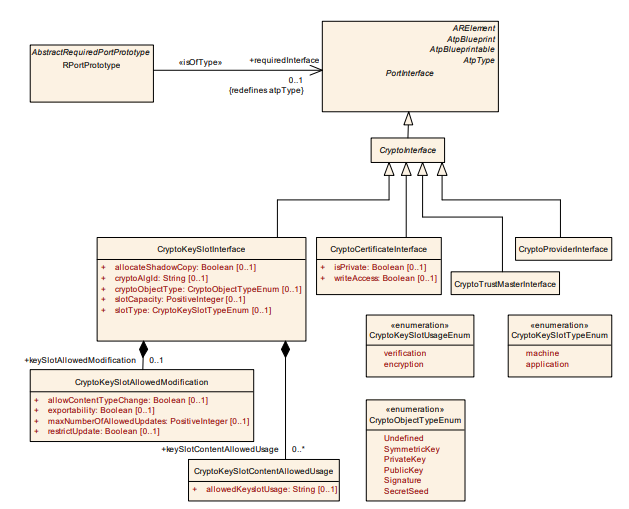
Below figure gives an example for the use of machine and application key slots.



## **Application Level and System Service Level**

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

Below figure shows the model elements that are used to configure access from an Adaptive Application to elements of FC Crypto.



Some Adaptive Platform Services such as update and configuration, communication, persistency or diagnostics also require cryptographic services as part of their functionality. If key-material is needed and must be configurable by an Adaptive Application (e.g. OEM key manager), the platform shall specify a Key Slot of slotType machine. To manage the key material a dedicated Adaptive Application (key-manager) may specify the same Key Slot (i.e. same parameters and slotType machine). During Integration this machine type key-slot resource must be linked to the key-manager.

# **Crypto Provider**

A Crypto Provider is a structural element that organizes cryptographic primitives. Every Crypto Provider represents exactly either one hardware element, e.g., trusted platform module (TPM) or hardware security module (HSM), or one software element, e.g., cryptographic library. When the systems provide multiple hardware elements and or software elements, then the same number of Crypto Providers exists as hardware and software elements are in the system.

Each Crypto Provider implements its supporting cryptographic primitives and is represented by an instance of CryptoProvider and identified during runtime via call to LoadCryptoProvider with InstanceSpecifier as an input parameter.

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

CryptoProvider provides multiple cryptography techniques implementation listed below:

* Random Number Generator (RNG)
* Hashing
* Message Authentication Code (MAC)
* Symmetric encryption(Block Cipher, Stream Cipher)
* Authenticated Encryption
* Key Wrapping
* Digital signatures
* Asymmetric encryption
* Key Encapsulation Mechanism (KEM)
* Key Exchange Protocol, Key Exchange Mechanism, and Key Exchange Scheme

Each CryptoProvider may provide zero, one, or more Random Number Generator (RNG) implementations. Therefore, the FC Crypto provides RandomGeneratorCtx context. The CryptoAPI provides the CreateRandomGeneratorCtx to generate this context. Application calls CreateRandomGeneratorCtx() to get handler to access RandomGeneratorCtx, later uses methods provided by RandomGeneratorCtx to perform operations(RNG). Similarly to access context and perform different operations, application needs to call respective methods of Hashing, MAC etc.

# **Key Storage Provider**

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

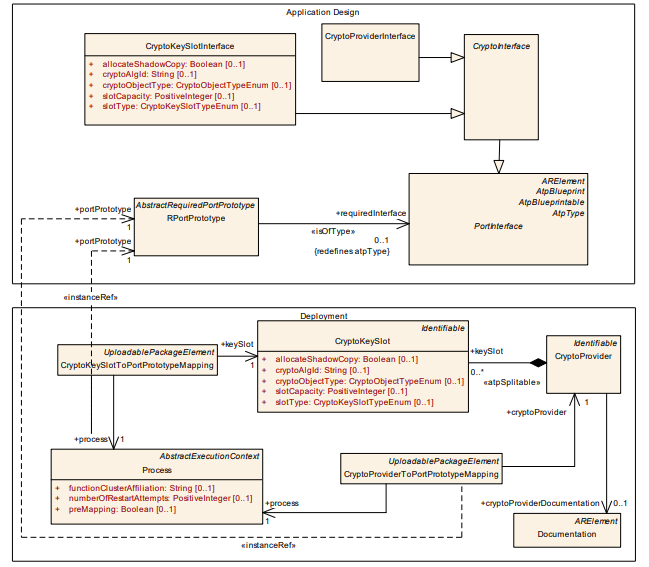
KeySlots used by application are defined by the integrator in the manifest via CryptoKeySlotInterface and CryptoKeySlotToPortPrototypeMapping.

FC Crypto provides access to the CryptoKeySlots for every AdaptiveApplicationSwComponentType. Every CryptoKeySlot is represented by RPortPrototype typed by CryptoKeySlotInterface in application design.

Assignment of CryptoKeySlots to a CryptoProvider is described in the manifest. So with the usage of a RPortPrototype that is typed by a CryptoKeySlotInterface the assignment to CryptoProvider is established.

The CryptoAPI provides a function to obtain CryptoProvider and with a call of MyProvider the FC Crypto provides the correspinding CryptoProvider of a KeySlot.

Below figure describes Key deployment:



The KeySlot identified during runtime. The CryptoAPI provides an interface with a call to LoadKeySlot to support this. The interface needs an InstanceSpecifier as an input parameter. Here, InstanceSpecifier represents a path to RPortPrototype mapped to needed CryptoKeySlot.

## **Serializable interface**

The CryptoAPI provides an interface for exporting of any public (by nature) objects, where additional integrity or confidentiality protection are not needed. Interfaces of all public (non-confidential) cryptographic objects and certificates that principally support serialization in plain (non-encrypted and non-authenticated) form are derived from the crypto::Serializable interface. Actually, this interface provides only one serialization method ExportPublicly.

## **Key** **Generation**

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

* **Symmetric Algorithms**: A symmetric system consists of a key, which is shared between the different parties.  
  The FC Crypto supports symmetric cryptography and allocate a new symmetric key object by a call of function GenerateSymmetricKey.
* **Asymmetric Algorithms**: Asymmetric systems consist of public and private key, which are generated. The public key is used for encryption, key encapsulation, or signature verification. The private key is used for decryption, key encapsulation, key exchange, or digital signature calculation.  
  The FC Crypto supports the asymmetric key generation and CrypotAPI provide such functionality. The private key is generated by calling GeneratePrivateKey.

# **Certificate handling (X.509 Provider)**

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

Applications or functional clusters can access certificates by CryptoCertificateInterface, which is provided by X.509 Provider.

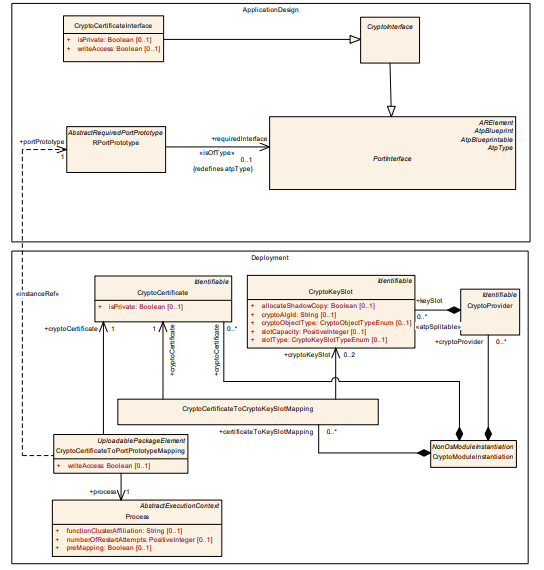
FC Crypto implementation includes a single X.509 Provider and responsibility of this provider is the support of Public Key Infrastructure (PKI).

A PKI contains a root certificate and one or many certificates. Main feature are:

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

The X.509 Provider stores and provide the root certificate and all needed CAs along the certification path, together with the reference to the corresponding public and private keys, which are handled by the Key Storage Provider.

Certificate deployment is shown in below figure:



# **Sequence Diagrams**

