VECTOR

MICROSAR Classic KeyM

Technical Reference

Version 4.11.01

Status	Released

Contents

Document Information

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_KeyManager.pdf	4.4.0
[2]	AUTOSAR	AUTOSAR_SWS_KeyManager.pdf	R21-11
[3]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.3.0
[4]	AUTOSAR	AUTOSAR_SWS_CryptoServiceManager.pdf	4.3.0
[5]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	4.4.0
[6]	AUTOSAR	AUTOSAR_SWS_NVRAMManager.pdf	4.0.3
[7]	IETF RFC5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	–
[8]	ITU-T X.690	ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)	Ed. 5 (08/2015)
[9]	IETF RFC2986	PKCS #10: Certification Request Syntax Specification	–
[10]	IETF RFC6960	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP	–
[11]	IETF RFC6961	The Transport Layer Security (TLS) Multiple Certificate Status Request Extension	–
[12]	BSI	TR-03110 Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS token	2.21
[13]	IETF RFC5755	An Internet Attribute Certificate Profile for Authorization	–

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

1 Introduction

This document describes the functionality, integration, API and configuration of the AUTOSAR BSW module KeyM.

The KeyM is specified in AUTOSAR *[1]*.

Supported Configuration Variants:	pre-compile	
Vendor ID:	KeyM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	KeyM_VENDOR_ID	109 decimal (according to *[5]*)

The KeyM is separated into two sub modules, the crypto key submodule and the certificate submodule.

The crypto key submodule provides services to introduce or update pre-defined cryptographic key material.

The certificate submodule provides services for different operations on certificates. It allows to define and configure certificates in a hierarchical PKI structure with root, intermediate and target certificates. In this way, certificates can be stored and updated in permanent or temporary storage. Furthermore, the submodule provides services to verify individual certificates against already stored and provided certificates in a chain. Besides, the submodule allows to access certificate data as well as specific certificate elements and to verify their contents.

1.1 Architecture Overview

The next figure shows the interfaces to adjacent modules of the KeyM. These interfaces are described in *API Description*.

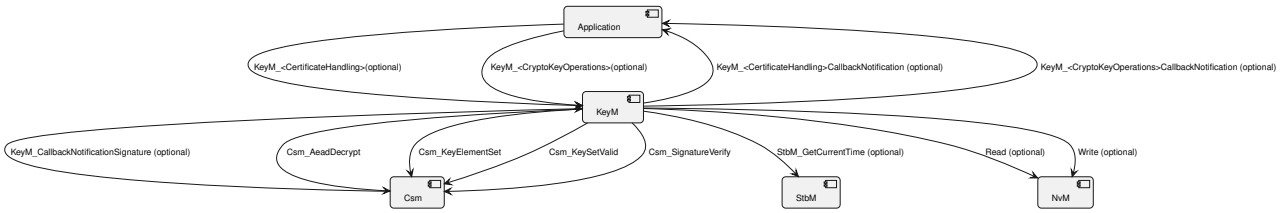

Figure 1.1 Interfaces to adjacent modules of the KeyM

2 Functional Description

2.1 Features

The features listed in the following tables cover the functionality specified for the KeyM. The AUTOSAR standard functionality is specified in *[1]*, the corresponding features are listed in the tables

> Supported AUTOSAR standard conform features

> Not supported AUTOSAR standard conform features

Vector Informatik provides further KeyM functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Features provided beyond the AUTOSAR standard

The following features specified in *[1]* are supported. Besides, the module is also compatible up to AUTOSAR version *[2]*.

Supported AUTOSAR Standard Conform Features
Set root and/or intermediate certificate
Update root and/or intermediate certificate
Set working certificate
Get certificate data from certificate slot in RAM or permanent key storage in CSM
Get certificate element
Get status of a certificate
Verify two certificates against each other
Verify certificate against certificate chain
Verify certificate against incomplete certificate chain
Retrieve iterated elements
Certificate validation period with StbM
Permanent certificate storage in NVM
Asynchronous CSM job processing
Certificate Signing Request (CSR)

Table 2.1 Supported AUTOSAR standard conform features

Supported Certificate Format Types
X.509 Public Key Infrastructure Certificate
Card Verifiable Certificate (CVC)
Certificate Revocation List (CRL)
Attribute Certificates (AC)

continues on next page

Table 2.2 – continued from previous page

Supported Certificate Format Types

Table 2.2 Supported Certificate Format Types

2.1.1 Deviations

The following features specified in *[1]* up to *[2]* and *[7]* are not supported:

Category	Description	Specification
Functional	Crypto Key Submodule	ASR 4.4.0 - R21-11
Functional	Key Generation for CSR	ASR 4.4.0 - R21-11
Functional	Security Events	ASR R21-11
Functional	Usage of KeyM_SetCertificate() for RAM certificates only	ASR R21-11
Functional	Key usage extension in CA certificates RFC 5280	ASR 4.4.0 - R21-11
Functional	Reset certificate status using KeyM_SetCertificate	ASR 4.4.0 - R21-11
Config	KEYM_E_CONFIG_FAILURE	ASR R21-11
Config	KeyMCertificateElementVerification	ASR 4.4.0 - R21-11
Config	KeyMCryptoKeyManagerEnabled	ASR 4.4.0 - R21-11
Config	KeyMCertificateManagerEnabled	ASR 4.4.0 - R21-11

Table 2.3 Not supported AUTOSAR standard conform features

The listed deviations are described in more detail in the following subchapters.

2.1.1.1 Crypto Key Submodule

Only services of the certificate handling sub module are currently supported. The list of services is described in chapter *Services provided by KeyM*.

2.1.1.2 Key Generation for CSR

Currently there is no support for asymmetric key generation within the module, which is required for the certificate signing request. In order to generate a CSR, the generated keys need to be set as CSM key elements in a previous step.

2.1.1.3 Security Events

Currently there is no support for security events reporting.

2.1.1.4 Usage of KeyM_SetCertificate for RAM certificates only

The API `KeyM_SetCertificate` is supported for all storage types (`KeyMCertificate`/ `KeyMCertificateStorage`.

2.1.1.5 Reset of certificate status using KeyM_SetCertificate

According to AUTOSAR, the API `KeyM_SetCertificate` can be used to reset the certificate status to `KEYM_CERTIFICATE_NOT_AVAILABLE`. In case of a certificate with configured initial value (see *Certificate Configuration*), the current certificate data in RAM is deleted and the initial value is loaded. The certificate status is set to `KEYM_CERTIFICATE_NOT_PARSED`.

2.1.1.6 KeyM_CertDataType

The data type KeyM_CertDataType is not supported according to *[2]*. Instead, KeyM supports the data type according to *[1]* with an uint8 pointer for `KeyM_CertDataPointerType`.

2.1.1.7 Key usage extension in CA certificates

According to *[7]* the key usage extension must be included in CA certificates that contain a public key that is used to validate other certificates. KeyM only verifies the key usage extension if it is included in the certificate, but does not insist on its availability and handles this extension as optional. If however the application requires a dedicated check whether the key usage extension is available, the service `KeyM_GetCertKeyUsage()` can be used for this purpose.

2.1.1.8 Development Error KEYM_E_CONFIG_FAILURE

The development error `KEYM_E_CONFIG_FAILURE` is currently not supported.

2.1.1.9 Configuration of element verification

Currently there is no support for configuring conditions and rules for the element verification. There is a workaround with an optional callout (see *Callout Functions*).

2.1.1.10 KeyMCryptoKeyManagerEnabled

Switch `KeyMCryptoKeyManagerEnabled` is provided but not used in implementation.

2.1.1.11 KeyMCertificateManagerEnabled

Switch `KeyMCertificateManagerEnabled` is provided but not used in implementation.

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Element Verification Callout
Public Key Setting Callout
Optional Certificate Elements
Optional Certificate Elements
Retrieve certificate identifier by name (see Chapter *KeyM_Cert_SearchCert*)
Retrieve iterable certificate elements by index (see Chapter *KeyM_CertificateElementGetByIndex*)
Dynamic issuer for certificates (see Chapter *Dynamic issuer and certificate groups*)
Dynamic certificate slot mapping (see Chapter *Dynamic issuer and certificate groups*)
Generic certificate revocation (see Chapter (see Chapter *Generic certificate revocation*))
Provide service to access certificate structures (see Chapter *Certificate structures*)
Provide service to access issuer certificate identifier
OCSP Stapling (see Chapter *OCSP Stapling*)
Remote handling of service requests (see Chapter *Remote Handling*)
OCSP response validity period (see Chapter *OCSP Response Validity Period*)

Table 2.4 Features provided beyond the AUTOSAR standard

2.1.3 Limitations

2.1.3.1 CSR

The current implementation supports only synchronous CSM jobs for the signature generation within the scope of a certificate signing request. In addition, only certificate signing requests for X.509 certificates are supported so far. For certificate signing requests with ECC public algorithm only the following ECC curves are supported:

> ED25519

> ED448

> SECP256

> SECP384

> SECP521

2.1.3.2 ECDSA certificates

The signature of an ECDSA certificate contains two integers, an r- and an s-component. If the lengths in bytes of these components are smaller than the maximum length according to the ECC curve type of the certificate, the KeyM will format the components in such a way that both are padded to the next larger multiple of 8 bytes with zero bytes. Thus, it must be ensured that the underlying Crypto modules can handle signatures formatted in this way.

2.1.3.3 RSA certificates

AUTOSAR does not specify a uniform structure for RSA keys. When KeyM sets the public key of an RSA certificate in the CSM, it does so by setting the key elements `CRYPTO_KE_CUSTOM_RSA_PUBLIC_EXPONENT` and `CRYPTO_KE_CUSTOM_RSA_MODULUS` in the target key. These non-standard key element identifiers are defined by all MICROSAR crypto drivers. If a non-MICROSAR crypto driver is used for RSA certificates, it needs to be ensured that either

> The used crypto driver can handle RSA keys with an exponent and a modulus element, and the above-mentioned preprocessor defines for the key element identifiers are set to the respective key element identifiers expected by the crypto driver (e.g., by manipulating Compiler_Cfg.h or some other header file included by KeyM)

or

> The RSA certificate in question defines a user-defined callout function for setting the public key (`/MICROSAR/KeyM/KeyMCertificate/KeyMCertificateSetKeyCalloutFunc`)

2.1.3.4 Remote Handling

The current implementation does not support remote handling for the services `KeyM_GetCertBasicConstraints()` and `KeyM_GetCertKeyUsage()`.

2.1.3.5 Subject Check

According to AUTOSAR *[1]* the subject field of the upper hierarchy shall match the issuer field of the certificate in the lower hierarchy. In contrast to the six-step string preparation algorithm mentioned in RFC-5280 *[7]* for comparison of the subject and issuer distinguished names (DNs), KeyM performs a simple binary comparison of the complete ASN.1 sequence. Consequently, any permutation of relative distinguished names (RDNs), absence of certain fields, or addition of extra fields will inevitably lead to a verification failure. In this case the certificate status will be set to `KEYM_CERTIFICATE_INVALID_CHAIN_OF_TRUST`.

2.1.3.6 Hashcalculation of subjectPublicKey value

The calculation for the hash over the public key is only supported for ECC certificates.

2.2 Initialization

Before any other functionality of the KeyM module can be called the initialization function `KeyM_Init()` has to be called by the BswM.

For manual null initialization of RAM variables, the KeyM offers the function `KeyM_InitMemory()` which can be called before the `KeyM_Init()`.

2.3 States

The certificate handling is split into smaller computational units that form a state machine. This way the main task of KeyM can perform asynchronously certificate operations as a background task.

The main function can handle only one certificate operation at a time. This operation may consist of multiple internal states. Internal states are:

- **Idle**: initial state before a service request
- **Init**: initialization of global RAM buffers for processing certificate
- **Parse**: parsing certificate data
- **Verify Elements**: verify parsed certificate elements
- **Subject Check**: compare issuer with subject of certificate in upper hierarchy
- **Time Stamp Check**: check certificate validation period
- **Set Key**: set public key in associated CSM key element
- **Verify Signature**: perform signature verification operation
- **Store**: store certificate data after successful verification
- **Notify**: notify application about end of service operation and verification result

The following sequence diagram depicts the states listed above using the example of setting a CA certificate. Furthermore, it gives an overview of the services provided by other BSW components used by the KeyM.

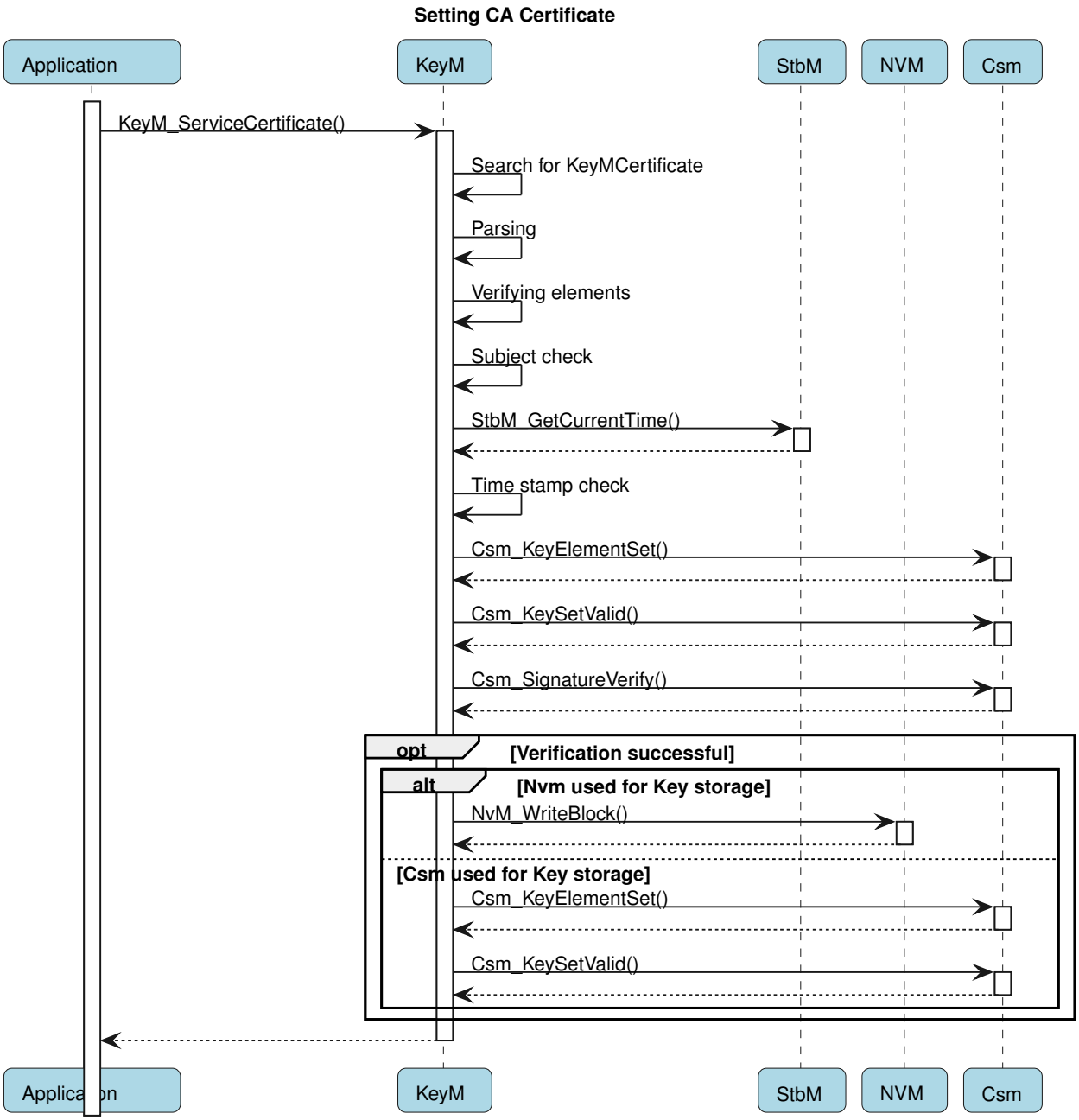

Figure 2.1 Setting CA Certificate

2.4 Certificate Status

The following diagram displays the different status of a certificate during a certificate verification operation. This status can be retrieved by calling the external API `KeyM_CertGetStatus()`. The numbers indicated at the state transitions show possible failure causes that can lead to this state transition.

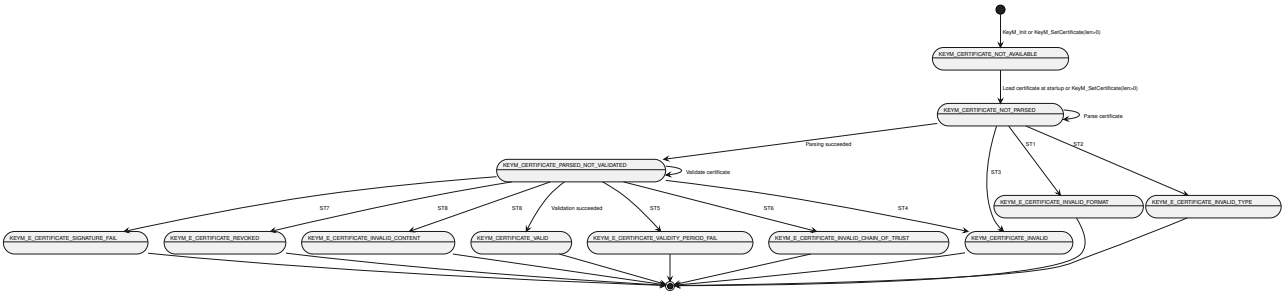

Figure 2.2 Certificate Status

State Transition Number	Possible cause
ST1	No valid ASN.1 format.
ST2	Certificate is not in a well-formatted form for X.509 or CVC.
ST3	Unspecified failure during parsing
ST4	Unspecified failure during validation
ST5	Current time stamp is not within the validity period of the certificate.
ST6	One of the configured upper certificates is missing or is not in a valid state (and could not be brought to this state during the validation). Optionally if certificate extension validation is enabled, this state indicates failure during the verification of the path length constraint value, contained in a basic constraints extension.
ST7	The signature verification of this certificate has failed or mismatched.
ST8	This certificate was identified on a revocation list maintained by the key manager.
ST9	Mismatch between the content of the certificate and the configured certificate elements.

Table 2.5 Failure State Transitions

2.5 Main Functions

The KeyM module offers two main functions. The first one is called cyclically and handles asynchronous jobs, whereas the second one can be used for background tasks. This is useful when it is called from a pre-emptive operating system when no other task operation is needed. The background main function can be used for calling time consuming synchronous functions.

2.6 Certificate Handling

This chapter gives an overview over the variety of general certificate operations which are provided by the certificate handling submodule of the KeyM.

2.6.1 ASN.1 Parser

The KeyM ASN.1 parser supports X.509 and CRL certificates *[7]*, CVC certificates *[12]* and AC certificates *[12]*. Each certificate element is encoded in tag, length, and value (TLV-) syntax according to DER rules *[8]*.

To perform any operation on a certificate, the content needs to be parsed fully in a previous step. Thereby after the parsing process one can retrieve a reference by accessing offset and length of each pre-configured element. The parsed information is stored in a temporary global RAM buffer. The parsing operation is performed whenever a certificate is set or updated within the function call of `KeyM_SetCertificate()`, `KeyM_ServiceCertificate()` or `KeyM_SetCertificateInGroup()`. Besides, when a certificate is being verified, all issuing certificates that are currently unparsed are parsed in the process. After parsing, a certificate's individual elements can be accessed via `KeyM_CertElementGet()`, `KeyM_CertElementGetFirst()` and `KeyM_CertElementGetNext()` (see Chapter *Services provided by KeyM*).

2.6.2 Element Verification

After a successful parsing, the data of each pre-configured certificate element is checked against specified rules and conditions. The configuration container `KeyMCertificateElementVerification` (see Chapter *Deviations*) specified by AUTOSAR is currently not supported. However, verification of certificate elements can be achieved by implementing the callout described in chapter *Appl_CertificateElementVerificationCallout*.

The verification of certificate elements is performed whenever a certificate is set or updated within the function call of `KeyM_SetCertificate()`, `KeyM_ServiceCertificate()` or `KeyM_SetCertificateInGroup()`.

2.6.3 Certificate Storage

The KeyM allows to configure certificates so that they can be stored at production time and further be used for several purposes. For example, root and intermediate certificates of a PKI system can be stored permanently in a specified certificate slot. If a certificate is presented to the ECU, this certificate can be stored in a temporary place to perform further verification.

2.6.3.1 Permanent Storage in CSM

The secure storage of certificates can be located in key storage locations of the CSM. To do this, one has to configure a corresponding CSM key and reference it in `KeyMCertificateCsmKeyTargetRef` within the configuration of the KeyM. For more detailed information on how to configure a CSM key, please refer to *[4]*.

2.6.3.2 Permanent Storage in NvM

The KeyM allows optional usage of NvM in order to store certificate data in non-volatile memory. To do so, the configuration container `KeyMNvBlock` needs to reference the corresponding NvM blocks. Validations in the DaVinci Configurator 5 ensure that the blocks in the NvM module are configured according to the KeyM's requirements.

The KeyM provides an optional feature to notify the NvM that a referenced memory block has changed (e.g. by calling `NvM_SetRamBlockStatus`). If the feature is disabled, the KeyM module does not mark a block as modified (`NvM_SetRamBlockStatus`), it is up to the NvM to detect the need of writing the block.

KeyM provides a callback for block initialization, reading from blocks and writing to blocks. All blocks need to be mapped to the `NvM_ReadAll` operation. The KeyM provides two modes for Block Processing:

- **DEFERRED**: The Block will only be marked as changed via `NvM_SetRamBlockStatus`.
- **IMMEDIATE**: The block is marked as changed via `NvM_SetRamBlockStatus` and the `NvM_WriteBlock` is called. It is possible to overwrite the NvM write function and configure it for NvM writing of KeyM. For this purpose, the name of the function must be entered in the configuration `/MICROSAR/KeyM/KeyMGeneral/KeyMNvWriteBlockFctName`. Therefore, there is a delay until the block is written to NvM.

Caution
Depending on the Block Processing mode, the KeyM tries to trigger the write operation. If the request to start the write operation fails (`NvM_SetRamBlockStatus` and `NvM_WriteBlock`), the KeyM service function will not return with an error. The KeyM will retry the operation in the next KeyM_MainFunction.
If the NvM operation fails and one of the configured callbacks reports an error, the write operation will not be retried. The failure needs to be detected by the customer using NvM. If an NVM write error occurs, the retry of the writing operation needs to be handled by the customer, e.g. by calling `NvM_WriteBlock` for the effected block.
Information on the NvM block status need to be retrieved via NvM.

Note
It is recommended to ensure that KeyM data, which are stored in one or more NvM blocks configured by `/MICROSAR/KeyM/KeyMCertificate/KeyMNvmBlock` or `/MICROSAR/KeyM/KeyMCRE/KeyMCRENvmBlock`, are restored unchanged. The data integrity can be ensured e.g. by NvM using CRC.

The KeyM module does handle its configured NvM blocks. Therefore, the KeyM provides a callback for block initialization, reading from blocks and writing to blocks. These callbacks are specified in *Callback Functions*. All blocks need to be mapped to the `NvM_ReadAll` operation and it is recommended to map DEFERRED blocks to the `NvM_WriteAll` operation.

Caution
Each `KeyM_NvBlock_Callback_KeyMCertificate_<NvBlock>` and `KeyM_NvBlock_Callback_CRE` must always be called for a write operation concerning `<NvBlock>`, this includes `NvM_WriteBlock` and `NvM_WriteAll`. This can be either ensured by configuration see *NvM Block Needs* or by user code.

2.6.4 Certificate Verification

The KeyM allows the verification of two certificates that are stored and parsed internally against each other. Thereby the KeyM references a pre-configured CSM job of the certificate Retrieving Certificate Data in the upper hierarchy that is used to verify the signature of the lower certificate (see Chapter *KeyM_VerifyCertificates*).

Furthermore, it is possible to verify a given certificate against all certificates in the associated certificate chain (see Chapter *KeyM_VerifyCertificate*). This chain can consist of certificates that are not permanently available according to the configuration, so that the KeyM also offers an additional service to pass the missing certificates to complete the chain for verification (see Chapter *KeyM_VerifyCertificateChain*).

2.6.5 Retrieving Certificate Data

The KeyM provides services to retrieve a complete certificate (see *KeyM_GetCertificate*) as well as specific certificate elements (see *KeyM_CertElementGet*, *KeyM_CertElementGetFirst*, *KeyM_CertElementGetNext*). Besides there is support for getting the current status of a given certificate (see *KeyM_CertGetStatus*).

Additionally, KeyM provides the service `KeyM_CertElementGetByStructureType` (see *KeyM_CertElementGetByStructureType*) to retrieve certificate elements out of certificate data passed as input by referencing a certificate structure type. The certificate structures that can be retrieved via this API are listed in Table *KeyM_CertificateStructureType*.

Due to ASN.1 formatting differences between different algorithms, the public key and the signature are retrieved as a constructed `BITSTRING` ASN.1 element containing the ASN.1 tag identifier and tag length. The rest of the certificate elements are retrieved as the plain tag value.

2.6.6 Service and verification notification to application

The KeyM invokes a callback notification to the application every time a requested service is processed. In addition, in case of a verification service, the result of the verification operation is conveyed. The provided interfaces are optional and have configurable function names (see Chapter *Notifications*).

2.6.7 Startup Handling

The KeyM provides a startup handling for previously stored certificates in permanent storage. Thereby after each startup, the stored certificates will be verified according to the configured hierarchy. Since the parsing and verification operation at startup can slow down the system, the startup handling for each certificate is optional. The custom enumeration parameter `KeyMCertStartUpHandling` in the configuration can be used to specify the startup behavior on a per-certificate basis. It allows parsing and verifying certain certificates (`PARSE_AND_VERIFY`). Furthermore, if the permanent certificate storage is unaltered, it is also possible to only parse and skip the verification operation (`PARSE_AND_TRUST`). However, the startup handling for a given certificate is disabled per default (`NONE`).

2.6.8 Certificate Update

The KeyM provides services for updating CA certificates (root and intermediate) as well as working certificates. When updating a CA certificate, the service first verifies the new certificate with the certificates in the upper hierarchies. In case of a new root certificate, the root is verified with its own public key (self-signed). After a successful verification and storage of the newly added certificate, the certificate status is set to valid, while all certificates in the lower certificate are invalidated automatically and their status is reset to parsed but not validated.

2.6.9 Certificate Revocation List

By adding a CRL, the KeyM allows to revoke certain certificates that are contained in the list. The revocation is handled right before a certificate signature can be verified. Since the CRL itself is verified by a CA certificate, it needs to be provided prior to any further verification of another working certificate in order to ensure a possible revocation.

A new CRL is installed into KeyM by calling `KeyM_ServiceCertificate` with the service `KEYM_SERVICE_CERTUPDATE_CRL`. During this installation, the certificate affected by the CRL is first set to the state `KEYM_CERTIFICATE_PARSED_NOT_VALIDATED`. A certificate is then set to state `KEYM_E_CERTIFICATE_REVOKED` during the next validation. This behavior is defined by AUTOSAR.

Note
During the installation of a CRL the parameter `revocationDate` is not checked.

2.6.10 Certificate Signing Request

In order to handle a certificate signing request, an asymmetric key pair needs to be generated. Both private and public key have to be set and validated as CSM key elements in advance. The public key shall be set in the CSM key reference `KeyMCertCsmSignatureVerifyKeyRef` of the corresponding certificate. Additionally, the private key shall be set in the key referenced by the configured CSM job for the signature generation in `KeyMCertCsmSignatureGenerateJobRef`. When setting the required key pair, it is possible to set the public key either as plain value or according to ASN.1 encoding rules including a leading zero byte as well as a format byte for ECDSA.

The necessary request data for a CSR, such as subject name and optional attributes, is initialized in a first step by calling `KeyM_InitCSR` and passing the request data in `CsrInfo`. For each array element in `CsrInfo` all configured subject names and as well optional attributes have to be set by initializing a data pointer, a data length and an element type before calling `KeyM_InitCSR`. All array elements need to be set in the exact order as the corresponding certificate elements are configured. This parameter points to an array of request data objects. The data returned by `KeyM_InitCSR` is then passed as request data in KeyM_ServiceCertificate for the service `KEYM_SERVICE_CERT_REQUEST_CSR`, which eventually generates the final CSR structure *[9]*. The advantage of this approach is that the complete set of CSR elements can be set with one function call of `KeyM_InitCSR`. However, this method does not support optional extensionRequest CSR elements.

Therefore, the KeyM supports a secondary iterative approach to set CSR element data. By calling `KeyM_CsrElementSet`, each CSR element can be set separately. In this regard, the certificate and element identifier as well as the encoding type need to be referenced. Since the configured elements are referenced by the element identifier, this approach brings the benefit that the order in which the CSR elements are set is no longer relevant. Besides, it is possible to set only a subset of the configured CSR elements, if for example certain elements are optional. While the element data for subject names and optional attributes has to be passed as plain data (`KEYM_CSR_ENCODING_NONE`), extensions need to be encoded (`KEYM_CSR_ENCODING_DER`) before being passed in `KeyM_CsrElementSet`. This input

data includes the complete DER encoded data that follows an object identifier in a certificate extension. After all CSR elements are set, calling `KeyM_ServiceCertificate` with the service `KEYM_SERVICE_CERT_REQUEST_CSR`, generates the final CSR structure *[9]*. Since no data is passed as input information for this request, the parameter RequestDataLength needs to be zero. Note, however, that `RequestDataLength` must still be a valid (i.e., not null) pointer.

The generated CSR structure is returned as response data of `KeyM_ServiceCertificate` and a service callback notification, which can be configured in `KeyMServiceCertificateCallbackNotificationFunc`.

After the Certificate Authority has signed the generated CSR, the resulting certificate can be stored and validated by calling `KeyM_ServiceCertificate` with the service `KEYM_SERVICE_CERT_UPDATE_SIGNED_CSR`.

Caution
When generating a CSR with ECDSA public key algorithm, it must be ensured that the plain public key value is padded with zero bytes if the public key length is smaller than the maximum length.

Caution
The generation of CSR with ECC public key algorithm is only supported for the following ECC curves:
> `ED25519`
> `ED448`
> `SECP256`
> `SECP384`
> `SECP521`

Caution
Only the following distinguished name attributes are supported for the CSR initialization with the services `KeyM_InitCSR` (deprecated) and KeyM_CsrElementSet:
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_COUNTRYNAME`
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_STATEORPROVINCENAME`
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_LOCALITYNAME`
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_ORGANIZATIONNAME`
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_ORGANIZATIONUNITNAME`
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_COMMONNAME`
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_EMAIL`
> `KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_DOMAINCOMPONENT`

2.6.11 Dynamic issuer and certificate groups

According to AUTOSAR *[1]*, the issuer for a given certificate must be configured statically within the PKI. However, some use-cases may require to flexibly add and remove certificates during the lifetime of an ECU. In order to meet this requirement, the MICROSAR Classic KeyM does not only support preset, statically configured issuers, but also offers a feature to determine certificates' issuers dynamically at runtime.

The concept of dynamic certificate issuers mainly has the advantage of flexible PKI handling, so that the user has to neither know the hierarchical relationships between certificates at configuration time nor keep a specific order in which certificates have to be installed. Furthermore, this approach saves resources, as the required memory for certificate data storage can be reduced, since it eliminates the need to keep empty and unused certificate slots for all possible configuration variants.

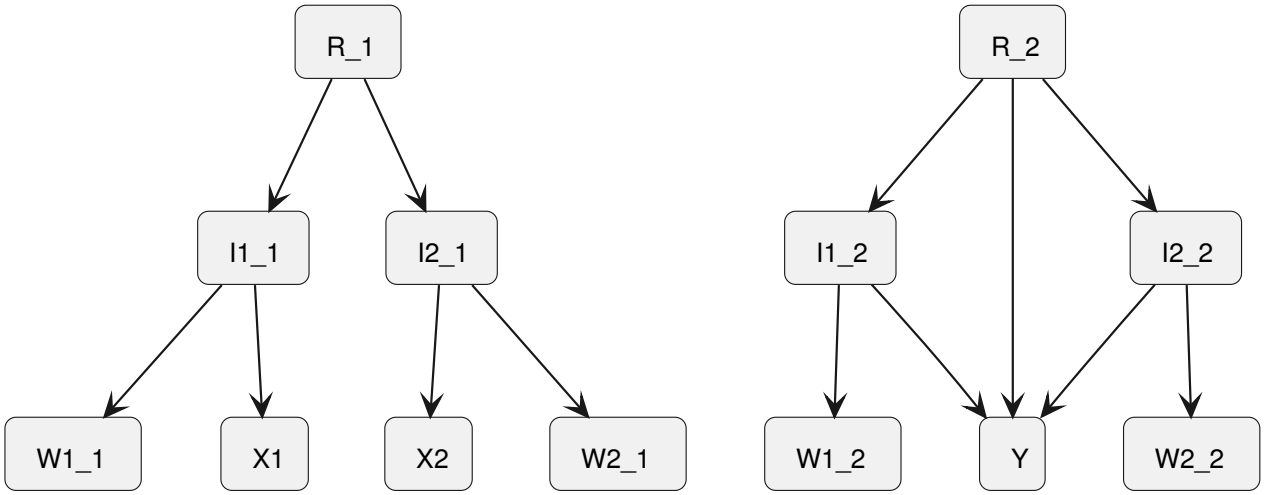

Figure 2.3 Preset Issuer vs. Dynamic Issuer

The figure above shows an example of a traditional preset-issuer scenario (left) and an example of a dynamic issuer scenario (right). Both examples depict the use-case in which a working certificate (named "X" and "Y" in the figure) can be issued by one of two intermediary CA certificates. In the traditional scenario, this use-case requires the user to create the working certificate twice, once with every possible issuer. In scenario 2, the working certificate is configured so that its issuer is determined dynamically at runtime. As shown in the figure, the configuration can be flexibly extended to even include the root certificate. In this scenario, no additional certificates are required and thus no additional memory resources are consumed.

If dynamic issuers are enabled for a certificate, its issuer is determined based on its issuer common name.

In theory, this may enable an attack in which an attacker installs a certificate with a forged subject common name, so that some certificates with dynamic issuers falsely identify it as their issuer. The attacker has thus tampered with the legitimate issuer relations within the PKI.

In order to thwart this kind of attack, the MICROSAR Classic KeyM introduces the concept of Certificate Groups.

Certificate Groups reference two kinds of certificates:

- **MEMBERS**, which may act as issuer for each other. Every certificate with a dynamic issuer must be a member of exactly one Certificate Group. All members of a Certificate Group must have the exact same structure (for reasons explained in the next paragraph), except for heterogeneous groups (see Chapter *Heterogeneous certificate groups*). Group members can be configured via a group's `KeyMCertificateGroupCertRef` parameter.
- **Additional Issuers**, which are themselves not members of the group but may act as issuer for group members. A certificate may act as an Additional Issuer for several Certificate Groups. This is illustrated in the figure below. Additional Issuers can be configured via a group's `KeyMCertificateGroupIssuerRef` parameter.

Besides defining which certificates may issue each other, Certificate Groups can also be used to dynamically determine slots for certificates at runtime. For example, the user may want to set a KeyM certificate at runtime, but does not know its ID. With Certificate Groups, all the user needs to know about the certificate is its Group ID. The KeyM will determine the certificate's slot based on the certificate's subject common name. If a certificate with the same subject name is already present in the group, the KeyM will update that certificate. If such a certificate is not yet present, it will select the next free slot within the group.

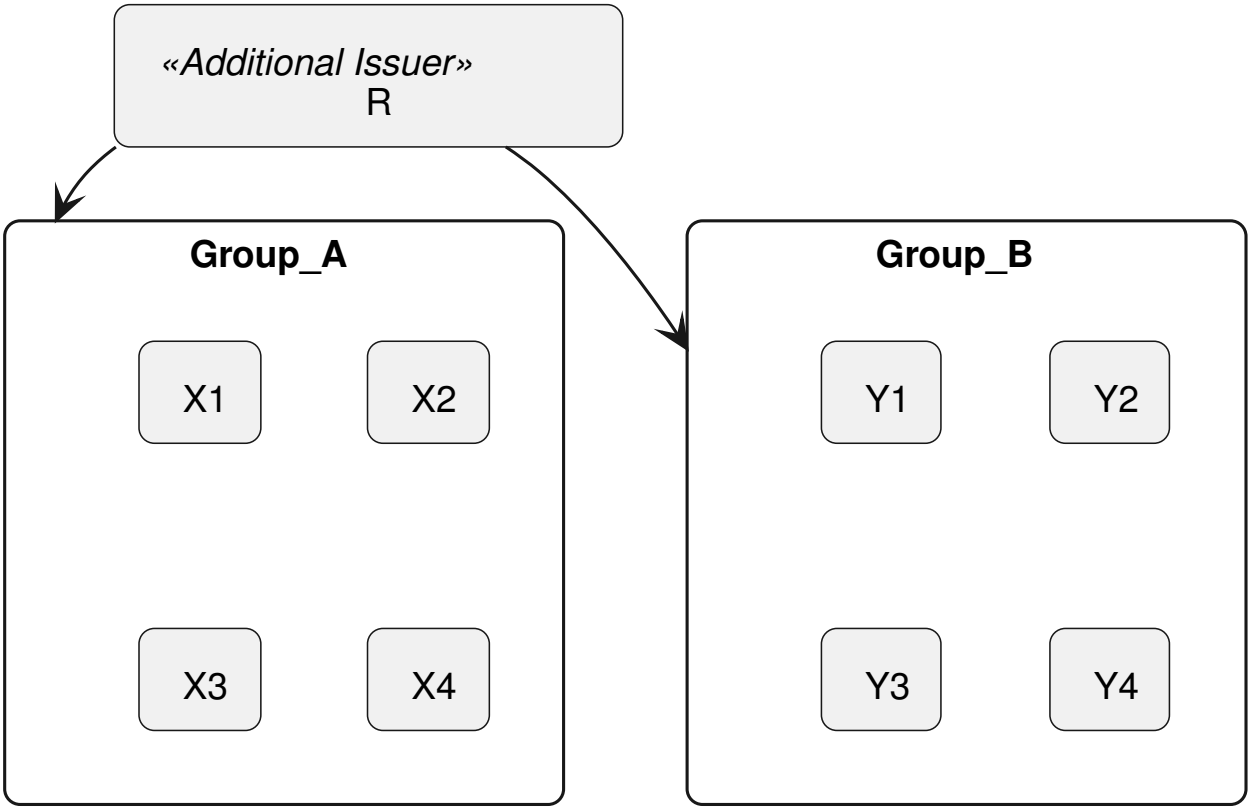

Figure 2.4 Group Certificates

There are three options for the configuration of a dynamic issuer:

- **DYNAMIC_MANDATORY_ISSUER** means that if no issuer with the respective subject common name is found, the certificate's status is set to invalid.
- **DYNAMIC_SELFSIGNED_OR_MANDATORY_ISSUER** works like the previous option, but also allows for the certificate to be self-signed.
- **DYNAMIC_OPTIONAL_ISSUER** means that the certificate is checked against its issuer if an issuer can be determined. If not, the verification is skipped and the certificate is deemed valid.

Members of a certificate group can be initially installed and updated using the service `KeyM_SetCertificateInGroup` (see 4.2.8.19). When a certificate is initially set, the parsing and the verification of certificate elements is processed synchronously within this function. When a certificate is updated all certificates in the lower hierarchies are invalidated

(Their certificate status is set to `KEYM_CERTIFICATE_PARSED_NOT_VALIDATE`) and a asynchronous verification against the upper certificate and whole lower chain is initiated. A certificate update requires the certificates to be installed in the chain order from top-to-bottom, since an update can be only performed if there is a valid issuer. After all certificates are installed initially, KeyM provides the possibility to verify all dynamic group certificates with one single service call using `KeyM_VerifyGroup` (see *KeyM_VerifyGroup*).

2.6.11.1 Heterogeneous certificate groups

Group member certificates need to be configured mostly in the same way. Nevertheless, it is possible to configure group member certificates with different algorithms within one certificate group. Certificates with the same algorithm form a sub-group within the group and need to be equally configured in-between. Note that the sub-groups are not visible in the configuration process. The algorithm of a configured certificate is specified by the algorithm family (`KeyMCertAlgorithmFamily`), algorithm type (`KeyMCertAlgorithmType`) and the configured OIDs (`KeyMCertificateElementObjectId`) of the certificate element for the public key algorithm info (structure type `CertificateSubjectPublicKeyInfo_PublicKeyAlgorithm`).

When certificate data is installed in a heterogeneous group with different algorithms, KeyM determines dynamically the certificate slot that matches the algorithm identifier specified in the certificate data.

2.6.12 Generic certificate revocation

Besides CRLs according to X.509, the KeyM also offers an additional generic certificate revocation concept. By setting so-called certificate revocation entries (CRE) which consist of an issuer common name and a certificate serial number, the certificate that is to be revoked can be uniquely identified. There is only a limited number of possible certificate revocation entries. This number is defined during the configuration stage. Certificates that have been revoked with CREs are revoked at runtime during the verification process. Every installed CRE is persisted to NvM and is read after each startup. All CREs are stored in a single NvM block.

For persisting the CREs in NvM, the same principles apply as for certificates. See *Permanent Storage in NvM* for further information.

2.6.13 Certificate structures

In addition to the `KeyM_CertElementGet` API (see *KeyM_CertElementGet*) defined by the AUTOSAR standard, the MICROSAR Classic KeyM offers an additional way to access certificate data on a more basic level.

The `KeyM_CertElementGet` API can return only the primitive ASN.1 data (i.e., the individual Integers, Bitstrings or UTFStrings, etc.) such as the contained elements in the certificate subject or public key sequences. In some cases, it may, however, be required to return whole ASN.1 structures, such as the whole certificate subject ASN.1 sequence with all its individual fields. The MICROSAR Classic KeyM offers such an API (see *KeyM_CertStructureGet*), which returns the whole structure together with its structure header. The certificate structures that can be retrieved via this API are listed in Table *KeyM_CertificateStructureType*.

There are some certificate elements that do not consist of ASN.1 structures, such as the Version or Serial Number fields. For these elements, the returned data consists only of the primitive element data.

2.6.14 Certificate hash

The KeyM offers the option to compute a hash over given certificate data. Using the optional configuration parameter `KeyMCertCsmHashJobRef`, one can specify a CSM hash job per certificate. By configuring the hash type using the enum parameter `KeyMCertHashType`, one can specify over which part of the certificate the hash is computed. The hash can be computed over the complete certificate data or over the subjectPublicKey value of the certificate (excluding the ASN.1 tag identifier, tag length and number of unused bits).

When configured, the corresponding hash is computed at startup for persisted certificates or synchronously after the certificate data is initially installed. By calling `KeyM_GetCertHash()` the computed hash can be retrieved (see 4.2.8.25). Using `KeyM_GetPubKeyHash()` the computed public key hash can be retrieved (see 4.2.8.26).

If the hash calculation fails during start up, the start up is not blocked but when the hash is retrieved, `KeyM_GetCertHash()` and `KeyM_GetPubKeyHash()` return E_NOT_OK. If hash calculation fails during the initial installation, the error E_NOT_OK is returned directly and the installation process aborted.

Note
It must be secured that in the CSM hash primitive, which is referenced in the CSM hash job, the result length is big enough to store the calculated hash.

2.6.15 OCSP Stapling

The Online Certificate Status Protocol (OCSP) is used to determine the revocation status of identified certificates *[10]*. By providing more timely revocation information, this method can be used as alternative or in addition to checking against a periodic CRL. OCSP overcomes the main limitation of CRLs: rather having to download and search through an entire CRL, the client can check the status of a single certificate thus reducing overhead and burden of the client. However, OCSP still requires the client to make requests to the CA which can result in a huge number of requests on the OCSP responder.

The Transport Layer Security (TLS) Extension framework offers clients to request the server's copy of the current status of certificates using a Certificate Status extension, which is also referred as OCSP stapling *[11]*. This method shifts the burden from the client to the server by reducing the number of roundtrips and network delays. The server makes the OCSP request to the OCSP responder and staples the OCSP responses to the certificates returned to the client. This allows the responses to be cached and then used multiple times for many clients.

Thus, a Certificate Status message is conveyed to the client, containing a list of all single OCSP responses. The KeyM offers a service to parse and verify a Certificate Status message and retrieve the revocation status for the corresponding certificates. For this purpose, the common AUTOSAR API `KeyM_ServiceCertificate()` (see Chapter *KeyM_ServiceCertificate*) can be used along with a custom service `KEYM_SERVICE_CERT_STATUS_OCSP`. The Certificate Status message is passed as input data in the parameters `RequestData` and `RequestDataLenght`. In case of an erroneous OCSP response within the Certificate Status message, the OCSP response status will be returned in `ResponseData` and `ResponseDataLength`. The remaining parameters are not used for the `KEYM_SERVICE_CERT_STATUS_OCSP` service but still need to be valid. Ensure that the parameter `CertNamePtr` is no null pointer. The parsing and verification of the passed Certificate Status message is processed synchronously.

The KeyM supports both status types for Certificate Status messages containing single or multiple OCSP responses. Besides, the KeyM is capable of processing OCSP responses of the `id-pkix-ocsp-basic` response type. Within this context, a single OCSP response can contain multiple certificates that are used to verify the OCSP signature. Furthermore, the KeyM offers full support of hashing algorithms for the CertId element. The used OCSP signature algorithm can be ECDSA or RSA.

After parsing of the Certificate Status message finished, the OCSP elements are verified according to *[10]*. The KeyM verifies the following:

- **OCSP response status**
- **Basic Response type**
- **Certificate identified in a received response**
- **Validity period of OCSP response (thisUpdate and nextUpdate)**
- **Version**
- **OCSP response signature**

> **Responder's signature against certificate chain**

> **Certificate Status Value**

Any other optional element within the OCSP response is parsed but not verified. If the OCSP response is valid and the certificate shall be revoked according to the Certificate Status Value, the KeyM will set the certificate status to `KEYM_E_CERTIFICATE_REVOKED` and a revocation entry is added (see Chapter *Generic certificate revocation*).

In order to enable the support of OCSP revocation for a given certificate, the optional sub container `KeyMOCSP` needs to be configured per certificate. If the OCSP response contains optional certificates for the OCSP signature verification, the optional parameter `KeyMOCSPDelegatedResponderRef` needs to be configured. The dynamic group referenced by this parameter is used to install the additional certificates. The hash algorithm used for the CertId element in a OCSP response can be specified by configuring `KeyMOCSPResponseCertIdHashCsmJobRef`. Furthermore, if the responder of the OCSP response is identified by a public key hash, instead of its distinguished name, the CSM job used for this hash operation can be configured in `KeyMOCSPResponderPubKeyHashCsmJobRef`. According to RFC6090, the used algorithm for the responder's public key hash shall be SHA1.

The KeyM sets certificate revocations entries (CREs) for each revoked certificate, which is why the CRE needs to be enabled in addition (see Chapter *Generic certificate revocation*).

The CSM Jobs referenced in `KeyMOCSPResponseCertIdHashCsmJobRef`, `KeyMOCSPResponderPubKeyHashCsmJobRef` and `KeyMCertCsmSignatureVerifyJobRef` need to be configured as synchronous, since the KeyM handles the service `KEYM_SERVICE_CERT_STATUS_OCSP` synchronously.

2.6.15.1 Retrieve OCSP Response Information

The KeyM provides an additional custom service `KeyM_CertInfoGet` (see *KeyM_CertInfoGet*), which can be used to access specific element data that is contained within a received OCSP response. Currently the module supports OCSP response status, the OCSP certificate status and the OCSP thisUpdate timestamp (for more details see *[10]*). The corresponding supported certificate information types for this service call are depicted in table *KeyM_CertificateInfoType*.

2.6.15.2 OCSP Response Validity Period

According to RFC 6960 *[10]*, OCSP responses must contain a time field *thisUpdate*, which is the most recent time at which the indicated certificate status is known to be correct by the OCSP responder. Furthermore, OCSP responses can optionally contain a time field *nextUpdate*, which is the time at or before which newer information will be available about the status of the certificate. If *nextUpdate* is not present, it is not possible to define a finite validity period for OCSP responses.

Therefore, MICROSAR KeyM provides a configuration parameter to define a maximum validity period for OCSP responses named *OCSP Response Validity Period*. If configured, the KeyM checks if the current time is within the time period defined by the OCSP response's *thisUpdate* time and the configured validity period.

If *nextUpdate* time is present and is before the configured OCSP Response Validity Period, the validity period is defined by *nextUpdate* time.

If the parameter is not instanciated, KeyM only checks if the current time is greater than the *thisUpdate* time and optionally if the current time is smaller than the *nextUpdate* time.

Caution
The OCSP Response Validity Period must be configured longer than the time interval between two OCSP responses sent by the Server. Otherwise, a valid OCSP response could be rejected.

2.6.16 Remote Handling

Incoming service requests to KeyM can be processed either on application side or alternatively on a remote instance (e.g. HSM). This is realized by transferring KeyM service requests through CSM jobs and key management APIs to a custom Crypto driver. This way the KeyM on application side acts solely as a proxy and any certificate operations are processed completely on remote side. The KeyM APIs remain the same, independently if the certificate is handled on application or remote side. In addition, the KeyM provides remote service dispatching functions (see *Dispatching Remote Service Requests*), that can be used on remote side. This has the advantage that serialization and deserialization of passed data is handled only within KeyM and the Crypto driver on remote side is independent from KeyM service requests.

In order to synchronize the application and remote side, a preconfiguration file is generated based on the certificate configuration on remote side. The preconfiguration contains the certificate configuration on application side. The preconfigured certificates require a CSM job reference with an AEAD Decrypt primitive. This CSM job as well as the CSM key referenced in the job are both used for the transfer of remote service requests. For this purpose the configuration parameter `KeyMCertCsmSignatureVerifyJobRef` and `KeyMCertCsmSignatureVerifyKeyRef` shall be used. Since the remote handling is processed synchronously, the used CSM jobs need to be synchronous as well.

Please note that certificates that are handled on the remote side display a slightly different callback behavior on the application side. Calls to `KeyM_ServiceCertificateCallbackNotification` (if configured) only report return codes `KEYM_RT_OK` and `KEYM_RT_NOT_OK` for remote certificates. Note that this is only a subset of the return codes that are used for local application certificates (see chapter *Appl_ServiceCallbackFunc*). Other, more detailed, return codes are simply reported as `KEYM_RT_NOT_OK`.

Caution
Please ensure that separate CSM jobs and CSM keys are configured for each certificate. The crypto key identifier in the referenced CSM job corresponds to the certificate identifier of the certificate on remote side. Therefore, the custom Crypto driver (e.g. Crypto_30_KeyM) shall provide a mapping between crypto key identifier and certificate identifier.
Also ensure that the used CSM job primitive is set to the AEAD Decrypt primitive provided by the custom Crypto driver.

Caution
Please consider that the startup handling for preconfigured certificates is processed on remote side only.

Caution
Since the revocation of a single certificate can have an effect on the complete PKI, CRE (see Chapter *Generic certificate revocation*) and OCSP (see Chapter *OCSP Stapling*) are processed either fully on application side or remote side. Mixed PKIs with application and remote certificates are suppressed by generator validation. This applies for both certificates with preset issuer as well as certificates with dynamic issuer (see Chapter *Dynamic issuer and certificate groups*).

2.6.16.1 Dispatching Remote Service Requests

The following table shows the provided dispatching functions and their corresponding service requests.

Dispatching Function	KeyM Service Request
KeyM_DispatchRemoteJob	KeyM_ServiceCertificate KeyM_ServiceCertificateById KeyM_VerifyCertificates KeyM_VerifyCertificate KeyM_VerifyCertificateChain KeyM_SetCertificateInGroup KeyM_VerifyGroup
KeyM_DispatchRemoteKeyElementSet	KeyM_SetCertificate KeyM_SetCRE KeyM_CsrElementSet KeyM_CertDelete

continues on next page

Table 2.6 – continued from previous page

Dispatching Function	KeyM Service Request
KeyM_DispatchRemoteKeyElementGet	KeyM_GetCertificate KeyM_CertElementGet KeyM_CertGetStatus KeyM_CertificateElementGetByIndex KeyM_CertificateElementGetCount KeyM_CertStructureGet KeyM_GetIssuerCertId KeyM_GetCertHash

Table 2.6 Dispatching Remote Service Requests

Caution
Due to parameter limitations of CSM primitives and key management APIs, the following KeyM services are not supported for remote service handling:
> `KeyM_CertElementGetFirst`
> `KeyM_CertElementGetNext`
> `KeyM_InitCSR`
> `KeyM_GetGroupCertId`

Caution
The current implementation does not provide remote service handling for the following services:
> `KeyM_GetCertBasicConstraints`
> `KeyM_GetCertKeyUsage`

2.6.17 Callback Notifications

There exist three different configurable callback notifications to provide information to the application about processing asynchronous service requests.

By configuring `KeyMServiceCertificateCallbackNotificationFunc` (see chapter *Appl_ServiceCallbackFunc*) the application is notified if a certificate service operation was finished and provides its status. A certificate service operation can be triggered by the APIs `KeyM_ServiceCertificate` or `KeyM_ServiceCertificateById`. This callback notification is called, if configured, only for the referenced certificate.

By configuring `KeyMCertificateVerifyCallbackNotificationFunc` (see chapter *Appl_VerifyCallbackFunc*) the application is notified if a verification operation was finished and provides its status. A verification operation can be triggered by the APIs `KeyM_VerifyCertificate`, `KeyM_VerifyCertificates`, `KeyM_VerifyCertificateChain`. This callback notification is called, if configured, for all certificates involved within the verification process.

By configuring KeyMCertificateGroupVerifyCallbackNotificationFunc (see chapter *Appl_VerifyGroupCallbackFunc*) the application is notified if a certificate group verification operation was finished and provides its status. A verification operation can be triggered by the API KeyM_VerifyGroup. This callback notification is called, if configured, for the referenced certificate group. The overall certificate group verification status is valid, only if all certificate group members could be verified successfully.

2.6.18 Certificate Slot Sharing

The KeyM provides the option to share certificate slots for certificate data and certificate status in between several certificates. This has the advantage that a series of certificate variants with different signature algorithms can be supported by only allocating one certificate slot. To achieve this, each certificate slot needs to be instantiated in KeyMCertificateSlot and can be referenced by the corresponding certificates in KeyMCertificateSlotRef.

Certificates with configured init values, can't be used for slot sharing.

Note
If a CA certificate shares a slot with another certificate and the CA certificate is preempted from the slot, all of its directly and indirectly issued certificates will remain in their current respective certificate statuses (e.g. KEYM_CERTIFICATE_VALID).
If a verification is not explicitly triggered on these certificates, they will remain in their certificate status until ECU restart and will, for example, not react to a revocation of an upper certificate.
If a verification is triggered on one of these certificates and the CA certificate is still preempted from the RAM slot, the verification will fail and the orphaned certificate's status will reflect this.

2.6.19 Certificate Deletion

The KeyM provides the option to delete installed certificates from temporary (RAM) and permanent storage (CSM, NvM). The certificate status of the deleted certificate is reset to KEYM_CERTIFICATE_NOT_AVAILABLE.

If a certificate with configured init value (see *Certificate Initial Value*) shall be deleted, the init value is reloaded into the certificate slot. The certificate status is set to KEYM_CERTIFICATE_NOT_PARSED.

In addition, all certificate related information (parse information, certificate's public key, hierarchical references, certificate slot references) is deleted or reset to the initial state. Besides, any valid certificate directly or indirectly issued by the deleted certificate is invalidated to the certificate status KEYM_CERTIFICATE_PARSED_NOT_VALIDATED.

The certificate deletion is processed synchronously using the service KeyM_CertDelete (see *KeyM_CertDelete*).

Note
Ensure that partial access is enabled for the referenced signature verify key reference (`Crypto_30_LibCv/Crypto/CryptoKeyElements/CryptoKeyElement/ CryptoKeyElementAllowPartialAccess`)
The service `KeyM_CertDelete` does not include the deletion of the private key in case of a CSR (see 2.6.10). The user must delete the private key manually.

2.6.20 Certificate Format Types

The KeyM supports the following certificate format types:

- X.509 - Public Key Infrastructure Certificate (according to RFC-5280 *[7]*)
- CVC - Card Verifiable Certificate (according to BSI TR-03110 *[12]*)
- CRL - Certificate Revocation List (according to RFC-5280 *[12]*)
- AC - Attribute Certificate (according to RFC-5755 *[12]*)

The certificate format type can be configured per certificate using the enumeration parameter `KeyMCertFormatType` in the configuration.

2.6.20.1 Attribute Certificates

Like other certificate format types, attribute certificates (AC) can be configured with certificate format option AC using the enumeration parameter `KeyMCertFormatType`. Additional the certificate identifier of the AC's holder certificate needs to be referenced in the configuration parameter `KeyMCertAttributeHolderCertRef`. The KeyM supports the parsing and verification of AC certificates. The installation of an AC certificate is covered with service `KeyM_SetCertificate` (see *KeyM_SetCertificate*).

Info
The verification of an AC certificate can be solely performed using the service `KeyM_VerifyCertificate` (see *KeyM_VerifyCertificate*). Other certificate verification services provided by the KeyM like `KeyM_VerifyCertificates`, `KeyM_VerifyCertificateChain` and `KeyM_VerifyGroup` do not support the certificate format type AC. Dynamic Group Handling is not supported for AC certificates at all.

Info

Due to the different options in the syntax of this particular certificate profile, certain AC elements can only be retrieved and configured as constructed and ASN.1 encoded strings. This is true for the AC's holder certificate (`KeyMCertificateElementObjectType`: 0x10) and AC's issuer certificate (`KeyMCertificateElementObjectType`: 0xA0). Also make sure to use the additional structure types for the holder (`KeyMCertificateElementOfStructure`: *CertificateAttributeHolder*) and certificate attributes (`KeyMCertificateElementOfStructure`: *CertificateAttribute*).

Caution

According to RFC-5755 (see *[13]*) the verification of an AC contains the verification of the complete AC's holder certification path as well as the complete AC's issuer certification path. Besides AC's period validity shall be checked. This is fully covered by KeyM. Any additional verification steps of optional elements (e.g. extensions, targeting check) are not covered per default. Nevertheless, the module allows the configuration of optional element verification callout to provide additional checks (see *Appl_CertificateElementVerificationCallout*).

2.6.21 Certificate Extension Validation

The KeyM provides the option to validate X.509 certificate extensions according to RFC 5280 (see *[7]*). Currently, only the basic constraints and key usage extensions are supported. The key usage is validated with a single deviation according to RFC 5280 (see *Key usage extension in CA certificates*). The validation of the extension data is performed when a certificate is installed and verified. This feature is enabled per default and can be disabled in the configuration (`/MICROSAR/KeyM/KeyMGeneral/KeyMCertificateExtensionValidationEnabled`).

By enabling this feature, KeyM provides additionally the services `KeyM_GetCertBasicConstraints()` and `KeyM_GetCertKeyUsage()` to retrieve extension data and perform further extension validation if required. The extension data is available for retrieval as soon as the certificate is parsed (certificate status is set to `KEYM_CERTIFICATE_PARSED_NOT_VALIDATED`).

2.6.22 Concurrent Service Requests

KeyM allows the processing of synchronous read access services during an already triggered and ongoing asynchronous service. This facilitates the retrieval of certificate data throughout the certificate life cycle, independently of any certificate operation that is still in progress. By requesting the current certificate status before any certificate data can be retrieved and copying the data inside an exclusive area, KeyM ensures that the certificate data is consistent.

This applies to following synchronous read access services:

> KeyM_GetGroupCertId

> KeyM_GetCertificate

> KeyM_CertElementGet

> KeyM_CertStructureGet

> KeyM_CertElementGetFirst

> KeyM_CertElementGetNext

> KeyM_CertificateElementGetByIndex

> KeyM_CertificateElementGetCount

> KeyM_GetCertHash

> KeyM_CertInfoGet

> KeyM_GetCertBasicConstraints

> KeyM_GetCertKeyUsage

In addition, KeyM allows the processing of synchronous write access services even during an already triggered and ongoing asynchronous service, as long as the exact certificate is not locked. Whenever a service writes certificate data, the corresponding certificate is locked, and other write access services cannot obtain the lock for this certificate. This facilitates the concurrent processing of services on different certificates.

This applies to following synchronous write access services:

> KeyM_SetCertificate

> KeyM_SetCertificateWithConstPtr

> KeyM_CsrElementSet

> KeyM_GetIssuerCertId

> KeyM_CertDelete

2.7 Client Server Interfaces

This implementation of the KeyM aims for supporting different AUTOSAR versions. The client server interfaces differ between the AUTOSAR versions. The main differences are typically the change of the parameter type or parameter order. Since the interfaces between the Client and the Server needs to be compatible but not identical, the KeyM can provide a dedicated interface for each AUTOSAR version introducing a change to the interface.

Please use the configuration parameter `/MICROSAR/KeyM/KeyMCertificate/KeyMCertificateSwcInterfaceVersion` to configure the required AUTOSAR version of the corresponding service port. Furthermore, this parameter gives a detailed description about the differences of the corresponding interfaces in the different AUTOSAR versions.

2.8 Error Handling

2.8.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in *[3]*, if development error reporting is enabled (i.e. pre-compile parameter `KeyM_DEV_ERROR_REPORT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured with the parameter `DetReportRuntimeErrorCallout`, but must have the same signature as the service `Det_ReportError()`.

The reported KeyM Module ID is 109. The reported service IDs identify the services which are described in *[1]*. The following table presents the service IDs and the related services:

Service ID	Service
0x01	KeyM_Init()
0x02	KeyM_Deinit()
0x03	KeyM_GetVersionInfo()
0x04	KeyM_Start()
0x05	KeyM_Prepare()
0x06	KeyM_Update()
0x07	KeyM_Finalize()
0x08	KeyM_Verify()
0x09	KeyM_ServiceCertificate()
0x0A	KeyM_SetCertificate()
0x0B	KeyM_GetCertificate()
0x0C	KeyM_VerifyCertificates()
0x0D	KeyM_VerifyCertificate()
0x0E	KeyM_VerifyCertificateChain()
0x0F	KeyM_CertElementGet()
0x10	KeyM_CertElementGetFirst()
0x11	KeyM_CertElementGetNext()
0x12	KeyM_CertGetStatus()
0x13	KeyM_ServiceCertificateByCertId()
0x19	KeyM_MainFunction()
0x1A	KeyM_MainBackgroundFunction()
0x80	KeyM_NvBlock_ReadFromBlock()
0x81	KeyM_NvBlock_WriteToBlock()
0x82	KeyM_NvBlock_Init()
0x83	KeyM_NvBlock_Callback()

continues on next page

Table 2.7 – continued from previous page

Service ID	Service
0x84	KeyM_NvBlock_ReadFromBlock_CRE()
0x85	KeyM_NvBlock_WriteToBlock_CRE()
0x86	KeyM_NvBlock_Init_CRE()
0x87	KeyM_NvBlock_Callback_CRE()
0x88	KeyM_CertificateElementGetByIndex()
0x89	KeyM_CertificateElementGetCount()
0x8A	KeyM_InitCSR()
0x8B	KeyM_ServiceCertificateById()
0x8C	KeyM_SetCertificateInGroup()
0x8D	KeyM_GetGroupCertId()
0x8E	KeyM_VerifyGroup()
0x8F	KeyM_SetCRE()
0x90	KeyM_CertStructureGet()
0x91	KeyM_GetIssuerCertId()
0x92	KeyM_GetCertHash()
0x93	KeyM_CsrElementSet()
0x94	KeyM_DispatchRemoteJob()
0x95	KeyM_DispatchRemoteKeyElementSet()
0x96	KeyM_DispatchRemoteKeyElementGet()
0x97	KeyM_SetCertificateWithConstPtr()
0x98	KeyM_VerifyCertificateChainWithConstPtr()
0x99	KeyM_Cert_SearchCert()
0x9A	KeyM_Cert_IsBusy()
0x9B	KeyM_CallbackNotificationSignature()
0x9C	KeyM_CertElementGetByStructureType()
0x9D	KeyM_CertDelete()
0x9E	KeyM_CertInfoGet()
0x9F	KeyM_GetCertBasicConstraints()
0xA0	KeyM_GetCertKeyUsage()
0xA1	KeyM_GetPubKeyHash()

Table 2.7 Service IDs

The errors reported to DET are described in the following table:

Error Code	Description
0x00	KEYM_E_NO_ERROR
0x01	KEYM_E_PARAM_POINTER
0x02	KEYM_E_SMALL_BUFFER
0x03	KEYM_E_UNINIT
0x04	KEYM_E_INIT_FAILED
0x80	KEYM_E_WRITE_ACCESS_FAILED
0x81	KEYM_E_CERTIFICATE_INIT_VALUE_ INVALID_LENGTH
0x82	KEYM_E_INVALID_CONFIGURATION

Table 2.8 Errors reported to DET

3 Integration

This chapter gives necessary information for the integration of the MICROSAR Classic KeyM into an application environment of an ECU.

3.1 Embedded Implementation

The delivery of the KeyM contains these source code files:

File Name	Description
KeyM.c	This is the main source file of the KeyM.
KeyM.h	This is the public main header file of the KeyM.
KeyM_Cert.c	This is the source file of the certificate handling sub module.
KeyM_Cert.h	This is the private header file of the certificate handling sub module.
KeyM_Asn1.c	This is the source file of the ASN.1 parser.
KeyM_Asn1.h	This is the private header file of the ASN.1 parser.
KeyM_Remote.c	This is the source file for remote service handling.
KeyM_Remote.h	This is the private header file for remote service handling.
KeyM_Utils.h	This is the private header file for common utility functions.
KeyM_Cbk.h	This is the public header file that contains declarations of callback functions for the CSM and the NVM.
KeyM_Cfg.c	This is the configuration source file.
KeyM_Cfg.h	This is the private configuration header file.
KeyM_Types.h	This is a public header for data types used for service interfaces of the KeyM.

Table 3.1 Source Code Files

3.2 Critical Sections

KeyM uses the following critical sections:

> **KEYM_EXCLUSIVE_AREA_0**

This critical section protects the main task busy state and ensures that the processing of the current main task is not interrupted. Furthermore, it ensures the consistency of the global RAM variables for the processing state, certificate Id, signature callback flag and signature callback result.

> **KEYM_EXCLUSIVE_AREA_1**

This critical section protects concurrent accesses to KeyM_CertStorage by KeyM and NVM, e.g. copy operations, and accesses to KeyM_NvBlock_State used for NVM handling. Furthermore, it ensures the consistency of global RAM variables for certificate data.

3.3 Certificate Configuration

A correct and complete certificate configuration is essential for the operation of the certificate handling submodule. Each certificate, including all its corresponding certificate elements, needs to be configured with respect to the plain certificate data as well as the hierarchical PKI. Several aspects of this are clarified in this chapter.

Caution
The configuration of the KeyM has dependencies to the Crypto Stack and the NvM. Therefore, it is necessary to always generate the KeyM in case the configuration of the Crypto Stack or NvM was changed.

3.3.1 Algorithm family

The `KeyMCertAlgorithmFamily` is a custom parameter and shall specify the required algorithm family for the signature verification operation of a given certificate. It is primarily necessary for setting the public key in the required format as a CSM key. This parameter needs to be set only for certificates with `KeyMCertAlgorithmType` set to ECC.

The following ECC public key algorithms are supported by KeyM:

- **ECDSA**
- **EDDSA**

By using an optionally configurable callout function for setting the public key (`/MICROSAR/KeyM/KeyMCertificate/KeyMCertificateSetKeyCalloutFunc`), further ECC curves can be supported.

3.3.2 Verification Job and Key Dependencies

The configuration of the `KeyMCertCsmSignatureVerifyJobRef` and `KeyMCertCsmSignatureVerifyKeyRef` must follow a strict pattern. While `KeyMCertCsmSignatureVerifyJobRef` references the CSM job that is used to verify the signature issued by the respective certificate, `KeyMCertCsmSignatureVerifyKeyRef` references the CSM key that is used to store the certificate's own public key. Root certificates are the only certificates that reference the same CSM job that is used for their verification as they are self-signed.

Caution
All signature verify jobs need to be configured either as synchronous or as asynchronous jobs. A mixed configuration leads to undefined behavior.

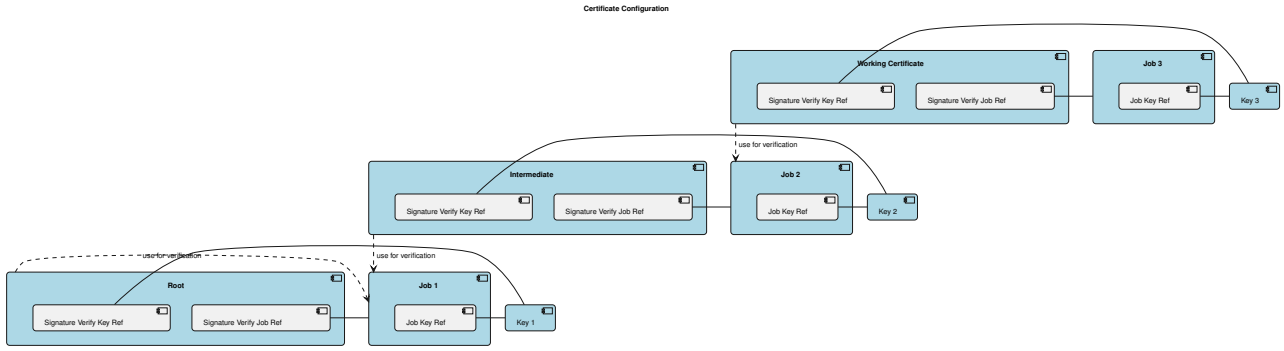

Figure 3.1 Certificate Verification Job and Key References

3.3.3 Certificate Initial Value

The optional configuration of the choice container `KeyMCertInit` for each certificate allows to define initial certificate data. By instantiating either a `KeyMCertInitValue` or `KeyMCertInitCallout` sub container, one can choose how the initial certificate data is provided.

The certificate initialization can be accomplished easily, by setting the corresponding certificate data in the `KeyMCertInitValue` configuration parameter. The data can be provided in the form of comma- or space-separated decimal or hexadecimal byte values (hexadecimal bytes should be prefixed with 0x). Alternatively, one can configure a callout in `KeyMCertInitCallout` which returns the certificate data and its data length.

Providing a certificate initial value is especially useful for CA (Certificate Authority) certificates, that are fixed and known by the manufacturer. This way the setting of a certificate during runtime is not necessary and can be achieved during the configuration phase. Nevertheless it is also possible to overwrite a given initial value by calling one of the set services, except for ROM certificates.

While for most certificate storage types (`KEYM_STORAGE_IN_RAM`, `KEYM_STORAGE_IN_CSM`, `KEYM_STORAGE_IN_NVM`) the configuration of a certificate initial value is optional, for read-only certificates (`KEYM_STORAGE_IN_ROM`) it is mandatory.

The configuration of a ROM certificate with an initial value has the advantage of reducing the necessary RAM memory which is used for each certificate slot.

Caution

Certificate information and the length of the certificate information provided via `KeyMCertInitCallout` may not be changed during the lifetime of the data. If this happens, the processing of the certificate information may return with errors.

Caution
In must be ensured, that the certificate configuration consists at least one certificate with storage type different from `KEYM_STORAGE_IN_ROM`. If this is not the case, a dummy certificate with storage type different from `KEYM_STORAGE_IN_ROM` (e.g. RAM) needs to be configured. The certificate max length for the dummy certificate can be reduced to a minimum, in order to safe storage.

3.3.4 Element configuration

Per default, it is required to configure only certificate elements that are either mandatory for the certificate verification (issuer name, validity period, subject name, public key, signature) or are of interest for the application and shall be retrievable after a certificate has been set.

By enabling the `KeyMCertAllowUnconfiguredElements` parameter per configured certificate, not all contained certificate elements need to be configured. Thus, the ASN.1 parser will accept certificate elements that are part of the certificate data but were not set at configuration stage. However, those certificate elements will just be skipped during the parsing and will not be available for further retrieving. If `KeyMCertAllowUnconfiguredElements` is disabled, all elements that are contained in the certificate data need to be configured accordingly.

Besides, the order in which the certificate elements are configured is per default flexible, so that the parsing is independent from any variations of the order within the distinguished names or extension elements for example. By enabling the `KeyMCertAllowFlexibleOrder` parameter per configured certificate, the order in which the certificate elements are configured does not need to be equal to the actual certificate data. If `KeyMCertAllowFlexibleOrder` is disabled, all certificate elements need to be configured in the exact order as they appear in the certificate data.

By configuring a certificate element as optional, it is ensured that if the element is present in the certificate data, it will be parsed and can be retrieved afterwards, but will not cause a parsing error if it is not present. This feature is useful especially for the configuration of optional elements within the extensions section of a certificate.

Furthermore, it is possible to replace configured certificate elements by enabling both the `KeyMCertAllowUnconfiguredElements` parameter for the given certificate and the `KeyMCertificateElementOptional` parameter for the given certificate element.

If there are several primitive ASN.1 sub-elements (e.g., Integer, Boolean) with the same tag identifier within a certificate extension that shall be retrievable afterwards, it is required to specify a certificate element path with the `KeyMCertificateElementPath` parameter. The element path can only be configured certificate extension elements.

This is used to specify the position of the configured certificate sub-element within its basic element structure. If no certificate element path is configured, the ASN.1 parser will match the first configured certificate element of the same tag identifier.

The path is based on the outer ASN.1 sequence element of the basic element structure. The certificate element path is configured in chapters so that each chapter corresponds to a

nesting level within the ASN.1 sequence. Chapters are specified with decimal integers and dots in between subchapters.

Exemplary configuration of the element path for the INTEGER within the following ASN.1 SEQUENCE and path 1.2.1:

```
SEQUENCE {
   OBJECT IDENTIFIER,
   OCTET STRING {
       INTEGER,
   }
}
```

Caution
If the certificate element path is applied to extensions that include optional sub-elements (e.g., criticality boolean flags), all configured sub-elements require an element path, including the optional subelements. If an optional sub-element is not available in the certificate data, KeyM dynamically adapts the statically configured element paths for the complete certificate extension.

Caution
Mandatory certificate elements that are relevant for the verification (e.g. public key, signature, validity period, issuer and subject common names) need to be configured in order to ensure an accurate processing.

3.3.5 Public key configuration

Within the scope of the configuration of a public key, the `KeyMCertificateElementOfStructure` parameter has to be set to `CertificateSubjectPublicKeyInfo_SubjectPublicKey` for the plain data of the public key element and `CertificateSubjectPublicKeyInfo_PublicKeyAlgorithm` for the algorithm object identifier.

3.3.6 Object Type

The `KeyMCertificateElementObjectType` parameter needs to be set according to the ASN.1 format. Control elements like Sequence and Set are constructed elements and therefore are not part of the configuration.

The following table shows the supported universal primitive ASN.1 tags by the parsing sub-module.

Element Description	Object Type
Boolean	0x01
Integer	0x02
Bit String	0x03
Octet String	0x04
NULL	0x05
Object Id	0x06
Enumerated	0x0A
UTF8	0x0C
Printable String	0x13
IA5 String	0x16
UTC Time	0x17
Generalized Time	0x18
BMPString	0x1E

Table 3.2 Supported universal ASN.1 tags

CVC certificates, in particular, include application elements with tags that are encoded in one or two octets. Besides the element class and structure type, the remaining tag number is an identifier for the element description type.

The table below shows the complete tag value for all supported element description types and the corresponding object type which should be used for the corresponding element in the configuration.

Element Description	Tag	Object Type
Certificate Profile Identifier	0x5F29	0x2
Certification Authority Reference	0x42	0xC
Public Key	0x7F49	0x81 / 0x86
Certificate Holder Reference	0x5F20	0xC
Certificate Holder Authorization Template	0x7F4C	0x4
Certificate Effective Date	0x5F25	0x17
Certificate Effective Date	0x5F25	0x17
Certificate Expiration Date	0x5F24	0x17
Signature	0x5F37	0x4
Discretionary Data	0x53	0x4

Table 3.3 Supported CVC Tags

The element for elliptic curve public keys can include all curve parameters marked with

corresponding tags (0x81 – 0x87). In this case the object type for the public key element has to be configured to 0x81.

It is also possible that only the public point is available as curve parameter in the plain data of the public key, designated with the tag 0x86. In this case the object type has to be configured to 0x86.

3.3.7 Configuration of CRLs

To configure a CRL in DaVinci configurator 5, a certificate with a special configuration needs to be considered. This certificate requires a certificate element with the parameter `KeyMCertificateElementOfStructure` set to the value `CertificateRevocationList` and the parameter `KeyMCertificateElementObjectType` set to the value “0x10”. Everything else can be configured like other certificates.

3.3.8 Configuration of Variant Time Formats

In order to support variant time formats (Generalized Time, UTC) for a time stamp element, only one certificate element must be configured. The object type can be configured to either 0x17 for UTC or 0x18 for GeneralizedTime. KeyM will accept both, no matter what the actual time format in the certificate data is.

If the certificate element for a given timestamp shall be also retrievable, it must be ensured, that the output buffer contains the maximum data length of both time formats, in this case 15 (UTC is 13 bytes long, GeneralizedTime is 15 bytes long).

3.4 Validation of Certificate Configuration

Configuring the KeyM module is not trivial and configuration errors can sometimes be hard to debug. For these reasons, the KeyM generator provides the option to test one’s current configuration of a `KeyMCertificate` container against a provided certificate. This validation can be triggered by instantiating the parameter `KeyMValidatorCertValue` and filling it with a valid certificate. The certificate can be provided in the form of comma- or space-separated decimal or hexadecimal byte values (hexadecimal bytes should be prefixed with 0x).

The validator will check the configuration parameters for certificate format, algorithm family, algorithm type and certificate max length on `KeyMCertificate` level. In addition, the certificate elements will be validated on `KeyMCertificateElement` level including the certificate element max length, the structure type, the object type, and the object identifier. The proposed certificate element max length is derived from the actual certificate element length. Therefore, it is possible that additional adjustment of this parameter is needed if different element lengths shall be supported.

If the supplied certificate contains elements that are not configured, the validator will suggest creating appropriate `KeyMCertificateElement` containers via solving actions. Especially when a certificate is configured for the first time, this feature can be very useful.

Caution
The validator currently only supports X509 certificates. Other formats, such as CVC or CRL, will produce a validation error.

Caution
For certificate extensions, the validator checks only the object identifier. Hence, when creating certificate elements for extensions via solving actions, the primitive ASN.1 elements of the extensions will be not included. These primitive ASN.1 elements need to be added manually if they are required and shall be retrievable.

Caution
When using the validator, make sure that the switches `KeyMCertAllowFlexibleElementOrder` and `KeyMCertAllowUnconfiguredElements` are enabled. Both switches are enabled by default.

4 API Description

4.1 Type Definitions

The types defined by the KeyM are described in this chapter.

4.1.1 KeyM_CertElementIteratorType

This structure is used to iterate through a number of elements of a certificate.

Struct Element Name	C-Type	Description	Value Range
certId	uint16	Holds the identifier of the certificate.	
offset	uint16	Holds the offset to the parsed element in the RAM buffer.	
elementLength	uint16	Holds the length of the element to be retrieved.	
rootElementIdx	uint16	Holds the element index of the root element.	
iterationStatus	uint8	Holds the current status of the iteration process.	> KEYM_CERT_ELEMENT_ITERATION_NOT_INITIALIZED > KEYM_CERT_ELEMENT_ITERATION_INITIALIZED > KEYM_CERT_ELEMENT_ITERATION_VALID > KEYM_CERT_ELEMENT_ITERATION_INVALID

Table 4.1 KeyM_CertElementIteratorType

4.1.2 KeyM_CSRInfoType

This structure is used to initialize the request objects for a CSR.

Struct Element Name	C-Type	Description	Value Range
dataPtr	uint8*	Points to an array that holds request object data.	
dataLength	uint16	Holds the length of the request object data.	

continues on next page

Table 4.2 – continued from previous page

Struct Element Name	C-Type	Description	Value Range
elementType	uint8	Defines the type of the request object.	> KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_COUNTRYNAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_STATEORPROVINCENAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_LOCALITYNAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_ORGANIZATIONNAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_ORGANIZATIONUNITNAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_COMMONNAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_SURNAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_SERIALNUMBER > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_STREETADDRESS > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_TITLE > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_GIVENNAME > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_EMAIL > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_USERID > KEYM_CERT_ASN1_CSR_ELEMENT_SUBJECT_DOMAINCOMPONENT > KEYM_CERT_ASN1_CSR_ELEMENT_ATTRIBUTE_UNSTRUCTUREDNAME

Table 4.2 – continued from previous page

Struct Element Name	C-Type	Description	Value Range

Table 4.2 KeyM_CSRInfoType

The buffer referenced by dataPtr must provide at least as many bytes as stored in dataLength.

4.1.3 KeyM_ConstCertDataType

This structure is used to provide initial certificate data with an optional callout (see *Appl_CertInitCallout*).

Struct Element Name	C-Type	Description
certData	KeyM_ ConstCertDataPointerType	Points to an array that holds initial certificate data.
certDataLength	uint32	Holds the length of the initial certificate data.

Table 4.3 KeyM_ConstCertDataType

The buffer referenced by certData must provide at least as many bytes as stored in certDataLength.

4.1.4 KeyM_ConstCertDataPointerType

This type is used in KeyM_ConstCertDataType.

Type Name	C-Type	Description
KeyM_ ConstCertDataPointerType	const uint8*	Points to an array that holds initial certificate data.

Table 4.4 KeyM_ConstCertDataPointerType

4.1.5 KeyM_CertificateGroupIdType

This type is used to identify a certificate group.

Type Name	C-Type	Description
KeyM_ CertificateGroupIdType	uint16	Holds the certificate group identifier.

Table 4.5 KeyM_CertificateGroupIdType

4.1.6 KeyM_CertificateGroupStatusType

This type is used for the overall status of a certificate group verification.

Type Name	C-Type	Description	Value Range
KeyM_CertificateGroup-StatusType	uint8	Holds the result of a certificate group verification.	KEYM_CERT_VERIFY_GROUP_VALID: All group member certificates were verified successfully. KEYM_CERT_VERIFY_GROUP_INVALID: One or more group member certificates could not be verified successfully.

Table 4.6 KeyM_CertificateGroupStatusType

4.1.7 KeyM_CertificateStructureType

This uint8-based enumeration type is used by APIs that allow the configuration-independent retrieval of certificate element data. There are multiple APIs like this and the way in which they format the retrieved data may differ slightly (see chapters *Retrieving Certificate Data* and *Certificate structures* and for details). Note that not all APIs are compatible with all structures. The right columns of the following table show which values are compatible with which API. Some values are currently not used by any API, but may be supported in the future.

Value	KeyM_CertStructureGet	KeyM_CertElementGet
KEYM_CERTIFICATE_ATTRIBUTE		
KEYM_CERTIFICATE_ATTRIBUTE_HOLDER		
KEYM_CERTIFICATE_EXTENSION	X	
KEYM_CERTIFICATE_ISSUER_NAME	X	
KEYM_CERTIFICATE_ISSUER_UNIQUE_IDENTIFIER		
KEYM_CERTIFICATE_REVOCATION_LIST		
KEYM_CERTIFICATE_SERIAL_NUMBER	X	X
KEYM_CERTIFICATE_SIGNATURE	X	X
KEYM_CERTIFICATE_SIGNATURE_ALGORITHM	X	X
KEYM_CERTIFICATE_SIGNATURE_ALGORITHM_ID	X	X
KEYM_CERTIFICATE_SUBJECT_NAME	X	

continues on next page

Table 4.7 – continued from previous page

Value	KeyM_CertStructureGet	KeyM_CertElementGet
KEYM_CERTIFICATE_SUBJECT_PUBLIC_KEY_INFO_PUBLIC_KEY_ALGORITHM		X
KEYM_CERTIFICATE_SUBJECT_PUBLIC_KEY_INFO_SUBJECT_PUBLIC_KEY		
KEYM_CERTIFICATE_SUBJECT_UNIQUE_IDENTIFIER		
KEYM_CERTIFICATE_VALIDITY_PERIOD_NOT_AFTER	X	
KEYM_CERTIFICATE_VALIDITY_PERIOD_NOT_BEFORE	X	
KEYM_CERTIFICATE_VERSION_NUMBER	X	X
KEYM_CERTIFICATE_SUBJECT_AUTHORIZATION		
KEYM_CERTIFICATE_SUBJECT_PUBLIC_KEY_INFO		X
KEYM_CERTIFICATE_VALIDITY_PERIOD		X
KEYM_CERTIFICATE_SUBJECT_PUBLIC_KEY_INFO_PUBLIC_KEY_ECC_CURVE		X
KEYM_REVOKED_CERTIFICATES		

Table 4.7 KeyM_CertificateStructureType

4.1.8 KeyM_CertificateInfoType

The certificate info type is used as input parameter in `KeyM_CertInfoGet()` to retrieve certificate related information.

Value	Description
OCSP Response Status	KEYM_CERT_INFO_OCSP_RESPONSE_STATUS
OCSP Cert Status	KEYM_CERT_INFO_OCSP_RESPONSE_CERT_STATUS
OCSP ThisUpdate Timestamp	KEYM_CERT_INFO_OCSP_RESPONSE_THISUPDATE

Table 4.8 KeyM_CertificateInfoType

4.1.9 KeyM_BasicConstraintsType

The basic constraints type is used as output parameter in `KeyM_GetCertBasicConstraints()` to retrieve certificate extension data (see *[7]*).

Struct Element Name	C-Type	Description
pathLenConstraint	uint32	Holds the maximum number of non-self-issued intermediate certificates that may follow this subject certificate in a valid certification path.
hasBasicConstraints	Boolean	Flag for availability of basic constraints extension.
isCritical	Boolean	Flag for criticality of the certificate extension.
isCA	Boolean	Flag indicates whether the subject certificate is a CA certificate.
hasPathLenConstraint	Boolean	Flag for availability of the optional pathLenConstraint value.

Table 4.9 KeyM_BasicConstraintsType

4.1.10 KeyM_KeyUsageType

The key usage type is used as output parameter in `KeyM_GetCertKeyUsage()` to retrieve certificate extension data (see *[7]*).

Struct Element Name	C-Type	Description
hasKeyUsage	Boolean	Flag for availability of key usage extension.
isCritical	Boolean	Flag for criticality of certificate extension.
digitalSignature	Boolean	Flag for purpose of public key in certificate.
nonRepudiation	Boolean	Flag for purpose of public key in certificate.
keyEncipherment	Boolean	Flag for purpose of public key in certificate.
dataEncipherment	Boolean	Flag for purpose of public key in certificate.
keyAgreement	Boolean	Flag for purpose of public key in certificate.
keyCertSign	Boolean	Flag for purpose of public key in certificate.
cRLSign	Boolean	Flag for purpose of public key in certificate.
encipherOnly	Boolean	Flag for purpose of public key in certificate.
decipherOnly	Boolean	Flag for purpose of public key in certificate.

Table 4.10 KeyM_KeyUsageType

4.2 Services provided by KeyM

4.2.1 KeyM_InitMemory

Prototype	
void **KeyM_InitMemory**( )	
Parameters	
None	
Return code	
void	
Functional Description	
The function initializes variables, which cannot be initialized with the startup code. Initialize component variables at power up.	
Limitations	
Module is uninitialized.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.2 KeyM_Init

Prototype	
void **KeyM_Init**(const KeyM_ConfigType * ConfigPtr)	
Parameters	
ConfigPtr	Pointer to the configuration set in VARIANT-POST-BUILD
Return code	
void	
Functional Description	
Initializes the Key Manager. This function initializes the KeyM module. It initializes all variables and sets the module state to initialized.	
Limitations	
Interrupts are disabled. Module is uninitialized.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.3 KeyM_Deinit

Prototype	
void **KeyM_Deinit**( )	
Parameters	
None	
Return code	
void	
Functional Description	
Resets the Key Manager. This function resets the KeyM module to the uninitialized state.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.4 KeyM_GetVersionInfo

Prototype	
void **KeyM_GetVersionInfo**(Std_VersionInfoType * VersionInfo)	
Parameters	
VersionInfo	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	
Functional Description	
Returns the version information. KeyM_GetVersionInfo() Returns version information, vendor ID and AUTOSAR module ID of the component.	
Limitations	
Configuration Variant(s): KEYM_VERSION_INFO_API == STD_ON	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.5 KeyM_MainFunction

Prototype	
void **KeyM_MainFunction**( )	
Parameters	
None	
Return code	
void	
Functional Description	
Main function of the module. Is called cyclically and handles asynchronous jobs.	
Limitations	
Declared and called by SchM.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.6 KeyM_MainBackgroundFunction

Prototype	
void **KeyM_MainBackgroundFunction**( )	
Parameters	
None	
Return code	
void	
Functional Description	
Main function for background tasks. Function is called from a pre-emptive operating system when no other task operation is needed. Can be used for calling time-consuming synchronous functions.	
Limitations	
Declared and called by SchM. Configuration Variant(s): KEYM_MAIN_BACKGROUND_FUNCTION_API == STD_ON	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.7 Key Sub-Module

4.2.7.1 KeyM_Prepare

Prototype	
Std_ReturnType **KeyM_Prepare**(const uint8* RequestData, uint32 RequestDataLength, uint8* ResponseData, uint32* ResponseDataLength)	
Parameters	
RequestData	Information that comes along with the request.
RequestDataLength	Length of data in the RequestData array.
ResponseData	Data returned by the function.
ResponseDataLength	In: Max number of bytes available in ResponseData. Out: Actual number.
Return code	
Std_ReturnType	E_OK: Service has been accepted and will be processed internally. Results will be provided through a callback. E_NOT_OK: Service not accepted due to an internal error. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with the expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Prepare key update. This function is used to prepare a key update operation. The main intent is to provide information for the key operation to the key server. Other operations may start the negotiation for a common secret that is used further to derive key material. This function is only available if KeyMCryptoKeyPrepareFunctionEnabled is set to TRUE.	
Limitations	
Configuration Variant(s): KEYM_CRYPTO_KEY_PREPARE_FUNCTION_ENABLED == STD_ON	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.7.2 KeyM_Start

Prototype	
Std_ReturnType **KeyM_Start**(KeyMStartType StartType, const uint8* RequestData, uint32 RequestDataLength, uint8* ResponseData, uint32 ResponseDataLength)	
Parameters	
StartType	Defines in which mode the key operation shall be executed.
RequestData	Information that comes along with the request, e.g., signature.
RequestDataLength	Length of data in the RequestData array.
ResponseData	Data returned by the function.
ResponseDataLength	In: Max number of bytes available in ResponseData. Out: Actual number.
Return code	
Std_ReturnType	E_OK: Start operation successfully performed. Key update operations are now allowed. E_NOT_OK: Start operation not accepted. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with the expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Starts a session for key operations. This function is optional and only used if the configuration item KeyMCryptoKeyStartFinalizeFunctionEnabled is set to true. It intends to allow key update operation.	
Limitations	
Configuration Variant(s): KEYM_CRYPTO_KEY_START_FINALIZE_FUNCTION_ENABLED == STD_ON	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.7.3 KeyM_Update

Prototype	
Std_ReturnType **KeyM_Update**(const uint8* KeyNamePtr, uint32 KeyNameLength, const uint8* RequestDataPtr, uint32 RequestDataLength, uint8* ResultDataPtr, uint32 ResultDataMaxLength)	
Parameters	
KeyNamePtr	Pointer to an array that defines the name of the key to be updated.
KeyNameLength	Specifies the number of bytes in keyName. The value 0 indicates that no keyName is provided within this function.
RequestDataPtr	Information that comes along with the request.
RequestDataLength	Length of data in the RequestData array.
ResultDataPtr	Pointer to a data buffer used by the function to store results.
ResultDataMaxLength	Max number of bytes available in ResultDataPtr.
Return code	
Std_ReturnType	E_OK: Service has been accepted and will be processed internally. Results will be provided through a callback. E_NOT_OK: Service not accepted due to an internal error. E_BUSY: Service could not be accepted because another operation is already ongoing. Try next time. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with the expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Initiate key update. This function is used to initiate the key generation or update process.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.7.4 KeyM_Finalize

Prototype	
Std_ReturnType **KeyM_Finalize**(const uint8* RequestDataPtr, uint32 RequestDataLength, uint8* ResponseDataPtr, uint32 ResponseMaxDataLength)	
Parameters	
RequestDataPtr	Information that comes along with the request.
RequestDataLength	Length of data in the RequestData array.
ResponseDataPtr	Data returned by the function.
ResponseMaxDataLength	In: Max number of bytes available in ResponseData. Out: Actual number of bytes in ResponseData or left untouched if the service runs in asynchronous mode and the function returns KEYM_E_OK.
Return code	
Std_ReturnType	E_OK: Service has been accepted and will be processed internally. Results will be provided through a callback. E_NOT_OK: Service not accepted due to an internal error. KEYM_E_BUSY: Validation cannot be performed yet. KeyM is currently busy with other jobs. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with the expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Finalize key update. The function is used to finalize key update operations. It is typically used in conjunction with the KeyM_Start operation and returns the key operation into the idle mode. Further key prepare or update operations are not accepted until a new KeyM_Start operation has been initialized. This function is only available if KeyMCryptoKeyStartFinalizeFunctionEnabled is set to TRUE. In addition, updated key material will be persisted and set into a valid state (calling Csm_KeySetValid).	
Limitations	
Configuration Variant(s): KEYM_CRYPTO_KEY_START_FINALIZE_FUNCTION_ENABLED == STD_ON	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.7.5 KeyM_Verify

Prototype	
Std_ReturnType **KeyM_Verify**(const AUTOMATIC* KeyNamePtr, uint32 KeyNameLength, const uint8* RequestData, uint32 RequestDataLength, uint8* ResponseData, uint32* ResponseDataLength)	
Parameters	
KeyNamePtr	Points to an array that defines the name of the key to be updated.
KeyNameLength	Specifies the number of bytes in KeyNamePtr. The value 0 indicates that no KeyNamePtr is provided within this function.
RequestData	Information that comes along with the request.
RequestDataLength	Length of data in the RequestData array.
ResponseData	Data returned by the function.
ResponseDataLength	In: Max number of bytes available in ResponseData. Out: Actual number of bytes in ResponseData or left untouched if the service runs in asynchronous mode and the function returns KEYM_E_PENDING.
Return code	
Std_ReturnType	E_OK: Operation was successfully performed. Result information are available. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_PENDING: Operation runs in asynchronous mode, has been accepted and will be processed internally. Results will be provided through callback. KEYM_E_BUSY: Validation cannot be performed yet. KeyM is currently busy with other jobs (for asynchronous mode). KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_KEY_CERT_INVALID: Key operation cannot be performed because the key name is invalid. KEYM_E_KEY_CERT_EMPTY: The key for this slot has not been set.
Functional Description	

continues on next page

Table 4.21 – continued from previous page

Verify key material. The key server requests to verify the provided keys. The key manager performs the operation on the assigned job and returns the result to the key server who verifies if the results were provided with this key as expected. This function is only available if KeyMCryptoKeyVerifyFunctionEnabled is set to TRUE.
Limitations
Configuration Variant(s): KEYM_CRYPTO_KEY_VERIFY_FUNCTION_ENABLED == STD_ON
Options
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE

4.2.8 Certificate Sub-Module

4.2.8.1 KeyM_ServiceCertificate

Prototype	
Std_ReturnType **KeyM_ServiceCertificate**(KeyM_ServiceCertificateType Service, const uint8* CertNamePtr, uint32 CertNameLength, const uint8* RequestData, uint32 RequestDataLength, uint32 ResponseDataLength, uint8* ResponseData)	
Parameters	
Service	Provides the type of service the key manager has to perform.
CertNamePtr	Points to an array that defines the name of the certificate to be updated.
CertNameLength	Specifies the number of bytes in CertNamePtr. The value 0 indicates that no CertNamePtr is provided within this function.
RequestData	Information that comes along with the request.
RequestDataLength	Length of data in the RequestData array.
ResponseDataLength	Max number of bytes available in ResponseDataPtr.
ResponseData	Data returned by the function.
Return code	

continues on next page

Table 4.22 – continued from previous page

Std_ReturnType	E_OK: Service data operation successfully accepted. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: Invalid chain of trust.
Functional Description	
Handle certificate operation requests. The key server requests an operation from the key client. The type of operation is specified in the first parameter KeyM_ServiceCertificateType. Certificate operation requests are operated through this function. This function is only available if the configuration parameter KeyMServiceCertificateFunctionEnabled is set to TRUE.	
Limitations	
RequestData and ResponseData must be valid pointers to user-provided buffers. Their respective length values must not exceed the actual buffer lengths. The parsing of a certificate and the verifying of certificate elements is performed synchronously within this function. Configuration Variant(s): KEYM_SERVICE_CERTIFICATE_FUNCTION_ENABLED == STD_ON	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.8.2 KeyM_SetCertificate

Prototype	
Std_ReturnType **KeyM_SetCertificate**(KeyM_CertificateIdType CertId, const KeyM_CertDataType* CertificateDataPtr)	
Parameters	
CertId	Holds the identifier of the certificate.
CertificateDataPtr	Pointer to a structure that provides the certificate data.
Return code	

continues on next page

Table 4.23 – continued from previous page

Std_ReturnType	E_OK: Certificate accepted. E_NOT_OK: Certificate could not be set. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Temporarily store certificate. This function provides the certificate data to the key management module to temporarily store the certificate.	
Limitations	
CertificateDataPtr->certData must be a valid, non-NULL pointer to a buffer of at least CertificateDataPtr->certDataLength bytes. The parsing of a certificate and the verifying of certificate elements is performed synchronously within this function.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.3 KeyM_SetCertificateWithConstPtr

Prototype	
Std_ReturnType **KeyM_SetCertificateWithConstPtr**(KeyM_CertificateIdType CertId, const KeyM_ConstCertDataType* CertificateDataPtr)	
Parameters	
CertId	Holds the identifier of the certificate.
CertificateDataPtr	Pointer to a structure that provides the certificate data.
Return code	
Std_ReturnType	E_OK: Certificate accepted. E_NOT_OK: Certificate could not be set. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Temporarily store certificate. This function is identical to KeyM_SetCertificate, but it accepts a const pointer to certificate data.	
Limitations	

continues on next page

Table 4.24 – continued from previous page

CertificateDataPtr->certData must be a valid, non-NULL pointer to a buffer of at least CertificateDataPtr->certDataLength bytes. The parsing of a certificate and the verifying of certificate elements is performed synchronously within this function.
Options
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE

4.2.8.4 KeyM_GetCertificate

Prototype	
Std_ReturnType **KeyM_GetCertificate**(KeyM_CertificateIdType CertId, KeyM_CertDataType* CertificateDataPtr)	
Parameters	
CertId	Holds the identifier of the certificate.
CertificateDataPtr	Provides a pointer to a certificate data structure. The buffer located by the pointer in the structure shall be provided by the caller of this function. The length information indicates the maximum length of the buffer when the function is called. If E_OK is returned, the length information indicates the actual length of the certificate data in the buffer.
Return code	
Std_ReturnType	E_OK: Certificate data available and provided. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_PARAMETER_MISMATCH: Certificate ID invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available, the certificate slot is empty. KEYM_E_KEY_CERT_READ_FAIL: Certificate cannot be provided, access denied.
Functional Description	
Provide certificate. This function provides the certificate data.	
Limitations	
CertificateDataPtr->certData must be a valid, non-NULL pointer to a buffer of at least CertificateDataPtr->certDataLength bytes.	

continues on next page

Table 4.25 – continued from previous page

Options
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE

4.2.8.5 KeyM_VerifyCertificate

Prototype	
Std_ReturnType **KeyM_VerifyCertificate**(KeyM_CertificateIdType CertId)	
Parameters	
CertId	Holds the identifier of the lower certificate in the chain.
Return code	
Std_ReturnType	E_OK: Certificate verification request accepted. Operation will be performed in the background and response is given through a callback. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_BUSY: Validation cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_PARAMETER_MISMATCH: Certificate ID invalid. KEYM_E_KEY_CERT_EMPTY: One of the certificate slots are empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: An upper certificate is not valid.
Functional Description	
Verify certificate. This function verifies a certificate that was previously provided with KeyM_SetCertificate() against already stored and provided certificates stored with other certificate IDs.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.8.6 KeyM_VerifyCertificates

Prototype	
Std_ReturnType **KeyM_VerifyCertificates**(KeyM_CertificateIdType CertId, KeyM_CertificateId-Type CertUpperId)	
Parameters	
CertId	Holds the identifier of the lower certificate in the chain.
CertUpperId	Holds the identifier of the upper certificate in the chain.
Return code	
Std_ReturnType	E_OK: Certificate verification request accepted. Operation will be performed in the background and response is given through a callback. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_BUSY: Validation cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_PARAMETER_MISMATCH: Certificate ID invalid. KEYM_E_KEY_CERT_EMPTY: One of the certificate slots are empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: An upper certificate is not valid.
Functional Description	
Verify two certificates. This function verifies two certificates that are stored and parsed internally against each other. The certificate referenced with CertId was signed by the certificate referenced with certUpperId. Only these two certificates are validated against each other.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.8.7 KeyM_VerifyCertificateChain

Prototype
Std_ReturnType **KeyM_VerifyCertificateChain**(KeyM_CertificateIdType CertId, const KeyM_CertDataType* certChainData, uint8 NumberOfCertificates)
Parameters

continues on next page

Table 4.28 – continued from previous page

CertId	Holds the identifier of the last certificate in the chain.
certChainData	This is a pointer to an array of certificates sorted according to the order in the PKI.
NumberOfCertificates	Defines the number of certificates stored in the certChainData array.
Return code	
Std_ReturnType	E_OK: Certificate verification request accepted. Operation will be performed in the background and response is given through a callback. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_BUSY: Validation cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_PARAMETER_MISMATCH: Certificate ID invalid. KEYM_E_KEY_CERT_EMPTY: One of the certificate slots are empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: An upper certificate is not valid.
Functional Description	
Verify list of certificates. This function performs a certificate verification against a list of certificates. It is a pre-requisite that the certificate that shall be checked has already been written with KeyM_SetCertificate() and that the root certificate is either in the list or is already assigned to one of the other certificates.	
Limitations	
NumberOfCertificates must be valid with respect to the configured certificate chain depth. certChainData must reference at least NumberOfCertificates many elements. For all its entries e, e->certData must be a valid, non-NULL pointer to a buffer of at least e->certDataLength bytes.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.8.8 KeyM_VerifyCertificateChainWithConstPtr

Prototype	
Std_ReturnType **KeyM_VerifyCertificateChainWithConstPtr**(KeyM_CertificateIdType CertId, const KeyM_ConstCertDataType* certChainData, uint8 NumberOfCertificates)	
Parameters	
CertId	Holds the identifier of the last certificate in the chain.
certChainData	This is a pointer to an array of certificates sorted according to the order in the PKI.
NumberOfCertificates	Defines the number of certificates stored in the certChainData array.
Return code	
Std_ReturnType	E_OK: Certificate verification request accepted. Operation will be performed in the background and response is given through a callback. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_BUSY: Validation cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_PARAMETER_MISMATCH: Certificate ID invalid. KEYM_E_KEY_CERT_EMPTY: One of the certificate slots are empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: An upper certificate is not valid.
Functional Description	
Verify list of certificates. This function is identical to KeyM_VerifyCertificateChain, but it accepts a const pointers to certificate data.	
Limitations	
NumberOfCertificates must be valid with respect to the configured certificate chain depth. certChainData must reference at least NumberOfCertificates many elements. For all its entries e, e->certData must be a valid, non-NULL pointer to a buffer of at least e->certDataLength bytes.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.8.9 KeyM_CertElementGet

Prototype	
Std_ReturnType **KeyM_CertElementGet**(KeyM_CertificateIdType CertId, KeyM_CertElementId-Type CertElementId, uint8* CertElementData, uint32* CertElementDataLength)	
Parameters	
CertId	Holds the identifier of the last certificate in the chain.
CertElementId	Specifies the ElementId where the data shall be read from.
CertElementData	Pointer to a data buffer allocated by the caller of this function. If available, the function returns E_OK and copies the data into this buffer.
CertElementDataLength	In: Pointer to a value that contains the maximum data length of the CertElementData buffer. Out: The data length will be overwritten with the actual length of data placed to the buffer if the function returns E_OK. Otherwise, the it will be overwritten with the value zero.
Return code	
Std_ReturnType	E_OK: Element found and data provided in the buffer. E_NOT_OK: Element data not found. KEYM_E_PARAMETER_MISMATCH: Certificate ID or certificate element ID invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate element too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available, the certificate slot is empty. KEYM_E_KEY_CERT_INVALID: The certificate is not valid or has not yet been verified.
Functional Description	
Provide certificate element. Provides the content of a specific certificate element. The certificate configuration defines how the certificate submodule can find the element, e.g. by providing the object identifier (OID). This function is used to retrieve this information if only one element is assigned to the respective OID.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.10 KeyM_CertElementGetFirst

Prototype	
Std_ReturnType **KeyM_CertElementGetFirst**(KeyM_CertificateIdType CertId, KeyM_CertElementIdType CertElementId, KeyM_CertElementIteratorType* CertElementIterator, uint32* CertElementDataLength)	
Parameters	
CertId	Holds the identifier of the last certificate in the chain.
CertElementId	Specifies the ElementId where the data shall be read from.
CertElementIterator	Pointer to a structure that is allocated and maintained by the caller. It shall not be destroyed or altered by the application until all elements have been retrieved through KeyM_CertElementGetNext().
CertElementDataLength	In: Pointer to a value that contains the maximum data length of the CertElementData buffer. Out: The data length will be overwritten with the actual length of data placed to the buffer if the function returns E_OK. Otherwise, the it will be overwritten with the value zero.
Return code	
Std_ReturnType	E_OK: Element found and data provided in the buffer. E_NOT_OK: Element data not found. KEYM_E_PARAMETER_MISMATCH: Certificate ID or certificate element ID invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate element too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available, the certificate slot is empty. KEYM_E_KEY_CERT_INVALID: Certificate is not valid or not verified successfully or referenced certificate element is not iterable.
Functional Description	
Provide first part of data from certificate element. This function is used to initialize the iterative extraction of a certificate data element. It always retrieves the top element from the configured certificate element and initializes the structure KeyM_CertElementIterator so that consecutive data from this element can be read with KeyM_CertElementGetNext().	
Options	

continues on next page

Table 4.31 – continued from previous page

Callcontext: TASK Reentrant: FALSE Synchronous: TRUE

4.2.8.11 KeyM_CertElementGetNext

Prototype	
Std_ReturnType **KeyM_CertElementGetNext**(KeyM_CertElementIteratorType* CertElementIterator, uint32* CertElementDataLength, uint8* CertElementData)	
Parameters	
CertElementIterator	Pointer to a structure that is allocated by the caller and used by the function. It shall not be destroyed or altered by the application until all elements have been read from the list.
CertElementDataLength	In: Pointer to a value that contains the maximum length of the CertElementData buffer. Out: The data length will be overwritten with the actual length of data placed to the buffer if the function returns E_OK.
CertElementData	Pointer to a data buffer allocated by the caller of this function. If available, the function returns E_OK and copies the data into this buffer.
Return code	
Std_ReturnType	E_OK: Element found and data provided in the buffer. The certElementIterator has been initialized accordingly. E_NOT_OK: Element data not found. KEYM_E_PARAMETER_MISMATCH: Certificate ID or certificate element ID invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate element too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available, the certificate slot is empty. KEYM_E_KEY_CERT_INVALID: Certificate is not valid or not verified successfully
Functional Description	
Provide further data from certificate element. This function provides further data from a certificate element, e.g. if a set of data is located in one certificate element that shall be read one after another. This function can only be called if the function KeyM_CertElementGetFirst() has been called once before. It has to be assured, that the installed certificate data remains consistent between calls of KeyM_CertElementGetFirst() and KeyM_CertElementGetNext as well as several calls of KeyM_CertElementGetNext().	

continues on next page

Table 4.32 – continued from previous page

Limitations
The passed CertElementIterator must be an object that was previously retrieved via KeyM_CertElementGetFirst().
Options
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE

4.2.8.12 KeyM_CertGetStatus

Prototype	
Std_ReturnType **KeyM_CertGetStatus**(KeyM_CertificateIdType CertId, KeyM_CertificateStatusType* Status)	
Parameters	
CertId	Holds the identifier of the certificate.
Status	Provides the status of the certificate.
Return code	
Std_ReturnType	E_OK: Certificate status available and provided. E_NOT_OK: Status provisioning currently not possible. KEYM_E_PARAMETER_MISMATCH: Certificate ID invalid.
Functional Description	
Provides certificate status. This function provides the status of a certificate.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.8.13 KeyM_Cert_SearchCert

Prototype	
boolean **KeyM_Cert_SearchCert**(const uint8* certNamePtr, uint32 certNameLength, KeyM_CertificateIdType* certId)	
Parameters	
certNamePtr	Pointer to a buffer that defines the name of the certificate.
certNameLength	Name buffer length.

continues on next page

Table 4.34 – continued from previous page

certId	Holds the identifier of the certificate.
Return code	
boolean	TRUE Certificate with given name is available. FALSE Certificate with given name is not available.
Functional Description	
Search for certificate name in configuration and return Cert ID if certificate was found.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.14 KeyM_CertificateElementGetByIndex

Prototype	
Std_ReturnType **KeyM_CertificateElementGetByIndex**(KeyM_CertificateIdType CertId, KeyM_CertElementIdType CertElementId, uint16 Index, uint8* CertElementData, uint32* CertDataLength)	
Parameters	
CertId	Holds the identifier of the certificate.
CertElementId	Holds the identifier of the iterable certificate element.
Index	This is the index to the respective element in the list of iterable elements.
CertElementData	Pointer to a data buffer for the iterable certificate element.
CertDataLength	In: Pointer to a value that contains the maximum data length of the CertElementData buffer. Out: The data length will be overwritten with the actual length of data placed to the buffer if the function returns E_OK.
Return code	

continues on next page

Table 4.35 – continued from previous page

Std_ReturnType	E_OK: Element found and data provided in the buffer. E_NOT_OK: Element data not found. KEYM_E_PARAMETER_MISMATCH: Certificate ID or certificate element ID invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate element too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available, the certificate slot is empty. KEYM_E_KEY_CERT_INVALID: Certificate is not valid or not verified successfully
Functional Description	
Provides data from an iterable certificate element.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.15 KeyM_CertificateElementGetCount

Prototype	
Std_ReturnType **KeyM_CertificateElementGetCount**(KeyM_CertificateIdType CertId, KeyM_CertElementIdType CertElementId, uint16* Count)	
Parameters	
CertId	Holds the identifier of the certificate.
CertElementId	Holds the identifier of the certificate element.
Count	Total number of iterable certificate elements.
Return code	
Std_ReturnType	E_OK: Element found and number of element items provided. E_NOT_OK: Element data not found. KEYM_E_PARAMETER_MISMATCH: Certificate ID or certificate element ID invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate element too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available, the certificate slot is empty. KEYM_E_KEY_CERT_INVALID: Certificate is not valid or not verified successfully.
Functional Description	

continues on next page

Table 4.36 – continued from previous page

Provides the amount of iterable elements.
Options
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE

4.2.8.16 KeyM_InitCSR

Prototype	
Std_ReturnType **KeyM_InitCSR**(const uint8* CertNamePtr, uint32 CertNameLength, const KeyM_CSRInfoType* CsrInfo, uint8 numOfReqObjects, uint8* ResponseData, uint32* ResponseDataLength)	
Parameters	
CertNamePtr	Points to an array that defines the name of the certificate.
CertNameLength	Specifies the number of bytes in CertNamePtr.
CsrInfo	Points to an array of request data objects.
numOfReqObjects	Total number of available request objects.
ResponseData	Data returned by the function.
ResponseDataLength	In: Max number of bytes available in ResponseData. Out: Actual number.
Return code	
Std_ReturnType	E_OK: CertificationRequestInfo data structure was generated successfully. E_NOT_OK: Due to internal error, the CertificationRequestInfo data structure could not be generated. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Initializes request data for certificate signing request.	
Limitations	
CsrInfo must reference at least numOfReqObject many elements.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.8.17 KeyM_ServiceCertificateById

Prototype	
Std_ReturnType **KeyM_ServiceCertificateById**(KeyM_ServiceCertificateType Service, KeyM_CertificateIdType CertId, const uint8* RequestData, uint32 RequestDataLength, uint8* ResponseData, uint32 ResponseDataLength)	
Parameters	
Service	Provides the type of service the key manager has to perform.
CertId	Holds the identifier of the certificate.
RequestData	Information that comes along with the request.
RequestDataLength	Length of data in the RequestData array.
ResponseData	Data returned by the function.
ResponseDataLength	Max number of bytes available in ResponseDataPtr.
Return code	
Std_ReturnType	E_OK: Service data operation successfully accepted. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: Invalid chain of trust.
Functional Description	
The key server requests an operation from the key client. The type of operation is specified in the parameter KeyM_ServiceCertificateType. Certificate operation requests are operated through this function. This function is only available if the configuration parameter KeyMServiceCertificateFunctionEnabled is set to TRUE. This function is identical to the function KeyM_ServiceCertificate(), but uses already the certificate identifier as parameter. In consequence there is no need to search the configured certificate by its name.	
Limitations	
RequestData and ResponseData must be valid pointers to user-provided buffers. Their respective length values must not exceed the actual buffer lengths. Configuration Variant(s): KEYM_SERVICE_CERTIFICATE_FUNCTION_ENABLED == STD_ON	

continues on next page

Table 4.38 – continued from previous page

Options
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE

4.2.8.18 KeyM_ServiceCertificateByCertId

Prototype	
Std_ReturnType **KeyM_ServiceCertificateByCertId**(KeyM_CertificateIdType CertId, KeyM_ServiceCertificateType Service, const uint8* RequestData, uint32 RequestDataLength, uint8* ResponseData, uint32 ResponseDataLength)	
Parameters	
CertId	Holds the identifier of the certificate.
Service	Provides the type of service the key manager has to perform.
RequestData	Information that comes along with the request.
RequestDataLength	Length of data in the RequestData array.
ResponseData	Data returned by the function.
ResponseDataLength	Max number of bytes available in ResponseDataPtr.
Return code	
Std_ReturnType	E_OK: Service data operation successfully accepted. E_NOT_OK: Operation not accepted due to an internal error. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: Invalid chain of trust.
Functional Description	

continues on next page

Table 4.39 – continued from previous page

The key server requests an operation from the key client. The type of operation is specified in the parameter KeyM_ServiceCertificateType. Certificate operation requests are operated through this function. This function is only available if the configuration parameter KeyMServiceCertificate-FunctionEnabled is set to TRUE. This function is identical to the function KeyM_ServiceCertificate-ById(), but uses already the certificate identifier as parameter. In consequence there is no need to search the configured certificate by its name. The functionality of this service is identifical to KeyM_ServiceCertificateById(). This service is specified in AUTOSAR R22-11.
Limitations
RequestData and ResponseData must be valid pointers to user-provided buffers. Their respective length values must not exceed the actual buffer lengths. Configuration Variant(s): KEYM_SERVICE_CERTIFICATE_FUNCTION_ENABLED == STD_ON
Options
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE

Note
The KeyM provides an API with the suffix *RteAdpt* in `KeyM_ServiceCertificateByIdRteAdpt`
This API is used only by RTE/SWCs. This API function wraps the existing API function KeyM_ServiceCertificateById and does not add functionality.

4.2.8.19 KeyM_SetCertificateInGroup

Prototype	
Std_ReturnType **KeyM_SetCertificateInGroup**(KeyM_CertificateGroupIdType GroupId, const uint8* RequestData, uint32 RequestDataLength, KeyM_CertificateIdType* CertId)	
Parameters	
GroupId	Holds the identifier of the certificate group.
RequestData	Pointer to the certificate data.
RequestDataLength	Holds the length of the certificate data.
CertId	Holds the certificate identifier of the slot where data has been installed.
Return code	

continues on next page

Table 4.40 – continued from previous page

Std_ReturnType	E_OK: Certificate accepted. E_NOT_OK: Certificate could not be set. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs or certificate is locked. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: Invalid chain of trust.
Functional Description	
Set group certificate. This function sets certificate data in a certicate group.	
Limitations	
This function call can trigger a callback notification if an optional service certificate callback <KeyM_ServiceCertificateCallbackNotification> is configured for the corresponding dynamic certificate slot. The parsing of a certificate and the verifying of certificate elements is performed synchronously within this function. If the certificate group contains a member certificate with the same subject name as the passed certificate data and the member certificate is currently revoked, an update service will return with E_NOT_OK.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.20 KeyM_GetGroupCertId

Prototype	
Std_ReturnType **KeyM_GetGroupCertId**(KeyM_CertificateGroupIdType GroupId, const uint8* SubjectCommonNameData, uint32 SubjectCommonNameDataLength, KeyM_CertificateIdType* CertId)	
Parameters	
GroupId	Holds the identifier of the certificate group.
SubjectCommonNameData	Pointer to the subject common name data.
SubjectCommonNameDataLength	Holds the length of the subject common name data.
CertId	Holds the certificate identifier of the slot where data has been installed.

continues on next page

Table 4.41 – continued from previous page

Return code	
Std_ReturnType	E_OK: Certificate identifier was successfully retrieved. E_NOT_OK: Referenced subject common name was not found within the group. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value.
Functional Description	
Get certificate identifier for previously set group certificates.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.8.21 KeyM_VerifyGroup

Prototype	
Std_ReturnType **KeyM_VerifyGroup**(KeyM_CertificateGroupIdType GroupId)	
Parameters	
GroupId	Holds the identifier of the certificate group.
Return code	
Std_ReturnType	E_OK: The verification of the certificate group was triggered successfully. E_NOT_OK: Certificate data is unavailable and no verification could be triggered. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs.
Functional Description	
Verify previously set group certificates.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.22 KeyM_SetCRE

Prototype	
Std_ReturnType **KeyM_SetCRE**(const uint8* IssuerNameData, uint16 IssuerNameDataLength, const uint8* SerialNumberData, uint16 SerialNumberDataLength)	
Parameters	
IssuerNameData	Points to an array that defines the issuer common name of the revoked certificate.
IssuerNameDataLength	Length of issuer common name data.
SerialNumberData	Points to an array that defines the serial number of the revoked certificate.
SerialNumberDataLength	Length of serial number data.
Return code	
Std_ReturnType	E_OK: Certificate revocation entry was appended successfully. E_NOT_OK: Due to internal error, the certificate revocation entry could not be appended. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Set a certificate revocation entry.	
Limitations	
Configuration Variant(s): KEYM_CRE == STD_ON	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: FALSE	

4.2.8.23 KeyM_CertStructureGet

Prototype	
Std_ReturnType **KeyM_CertStructureGet**(KeyM_CertificateIdType CertId, KeyM_CertificateStructureType CertStructure, uint8* CertStructureData, uint32* CertStructureDataLength)	
Parameters	
CertId	Holds the identifier of the certificate.
CertStructure	Holds the certificate structure type.
CertStructureData	Pointer to a valid buffer which will hold the data returned by the function.

continues on next page

Table 4.44 – continued from previous page

CertStructureDataLength	In: Max number of bytes available in CertStructureData. Out: Actual number.
Return code	
Std_ReturnType	E_OK: Certificate structure was retrieved successfully. E_NOT_OK: Due to internal error, the certificate structure could not be retrieved. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_KEY_CERT_INVALID: The certificate is not valid or has not yet been verified.
Functional Description	
Retrieve certificate structure.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.8.24 KeyM_GetIssuerCertId

Prototype	
Std_ReturnType **KeyM_GetIssuerCertId**(KeyM_CertificateIdType CertId, KeyM_CertificateId-Type* IssuerCertId)	
Parameters	
CertId	Holds the certificate identifier.
IssuerCertId	Holds the certificate identifier of the issuer this function returns.
Return code	
Std_ReturnType	E_OK: Issuer's certificate identifier was retrieved successfully. E_NOT_OK: Due to internal error, the issuer's certificate identifier could not be retrieved.
Functional Description	
Get certificate identifier of issuer in upper hierarchy.	
Options	

continues on next page

Table 4.45 – continued from previous page

Callcontext: TASK Reentrant: TRUE Synchronous: TRUE

4.2.8.25 KeyM_GetCertHash

Prototype	
Std_ReturnType **KeyM_GetCertHash**(KeyM_CertificateIdType CertId, uint8* HashData, uint32* HashDataLength)	
Parameters	
CertId	Holds the identifier of the certificate.
HashData	Pointer to a valid buffer which will hold the data returned by the function.
HashDataLength	In: Max number of bytes available in HashData. Out: Actual number.
Return code	
Std_ReturnType	E_OK: Certificate hash was retrieved successfully. E_NOT_OK: Due to internal error, the certificate hash could not be retrieved. KEYM_E_BUSY: Service cannot be performed yet. Certificate is locked by another service request. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_PARAMETER_MISMATCH: Parameter size doesn't match. KEYM_E_KEY_CERT_INVALID: Certificate is not valid. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty.
Functional Description	
Retrieve precomputed hash over certificate data.	
Limitations	
The current implementation does not support a certificate locking mechanism. Therefore the return code KEYM_E_BUSY is added only for future compliance. If a certificate update fails and the persisted certificate data is re-loaded, the hash needs to be computed again. It is possible that this function returns the hash of the invalid, updated certificate data if the hash computation has not finished yet. To ensure the validity of the hash, check the status of the updated certificate in advance. If after an update, the status is KEYM_CERTIFICATE_VALID, the hash retrieved by this function is valid.	
Options	

continues on next page

Table 4.46 – continued from previous page

Callcontext: TASK Reentrant: TRUE Synchronous: TRUE

4.2.8.26 KeyM_GetPubKeyHash

Prototype	
Std_ReturnType **KeyM_GetPubKeyHash**(KeyM_CertificateIdType CertId, uint8* HashData, uint32* HashDataLength)	
Parameters	
CertId	Holds the identifier of the certificate.
HashData	Pointer to a valid buffer which will hold the data returned by the function.
HashDataLength	In: Max number of bytes available in HashData. Out: Actual number.
Return code	
Std_ReturnType	E_OK: Public key hash was retrieved successfully. E_NOT_OK: Due to internal error, the public key hash could not be retrieved. KEYM_E_BUSY: Service cannot be performed yet. Certificate is locked by another service request. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_PARAMETER_MISMATCH: Parameter size doesn't match. KEYM_E_KEY_CERT_INVALID: Certificate is not valid. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty.
Functional Description	
Retrieve precomputed hash over public key of certificate. Note: The calculation for the hash over the public key is only supported for ECC certificates.	
Limitations	
The current implementation does not support a certificate locking mechanism. Therefore the return code KEYM_E_BUSY is added only for future compliance. If a certificate update fails and the persisted certificate data is re-loaded, the hash needs to be computed again. It is possible that this function returns the hash of the invalid, updated certificate data if the hash computation has not finished yet. To ensure the validity of the hash, check the status of the updated certificate in advance. If after an update, the status is KEYM_CERTIFICATE_VALID, the hash retrieved by this function is valid.	
Options	

continues on next page

Table 4.47 – continued from previous page

Callcontext: TASK Reentrant: TRUE Synchronous: TRUE

4.2.8.27 KeyM_CsrElementSet

Prototype	
Std_ReturnType **KeyM_CsrElementSet**(KeyM_CertificateIdType CertId, KeyM_CertElementId-Type CertElementId, KeyM_CsrEncodingType EncodingType, const uint8* ElementData, uint32 ElementDataLength)	
Parameters	
CertId	Holds the identifier of the certificate.
CertElementId	Holds the identifier of the certificate element.
EncodingType	Holds the encoding type of the certificate element.
ElementData	Points to an array of element data.
ElementDataLength	Max number of bytes available in ElementData.
Return code	
Std_ReturnType	E_OK: CSR element was set successfully. E_NOT_OK: Due to internal error, the CSR element could not be set. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Set certificate signing request element data.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

continues on next page

Table 4.49 – continued from previous page

4.2.8.28 KeyM_DispatchRemoteJob

Prototype	
Std_ReturnType **KeyM_DispatchRemoteJob**(const Crypto_JobType* job)	
Parameters	
job	Pointer to the configuration of the job which is owned by the API user. Contains structures with job and primitive relevant information but also pointers to result buffers.
Return code	
Std_ReturnType	E_OK: Remote service request was dispatched and processed successfully. E_NOT_OK: Due to an internal error, the remote service request could not be dispatched or processed. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty. KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: Invalid chain of trust.
Functional Description	
Dispatches remote Crypto job to KeyM service. Note: The dispatching functionality shall be only used by a custom Crypto driver on remote side to enable remote service handling.	
Limitations	
job->jobPrimitiveInputOutput->{input | secondaryInput | tertiaryInput | output}Ptr must be non-NULL, valid pointers and their respective length fields must be valid.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.8.29 KeyM_DispatchRemoteKeyElementSet

Prototype	
Std_ReturnType **KeyM_DispatchRemoteKeyElementSet**(uint32 cryptoKeyId, uint32 keyElementId, const uint8* keyPtr, uint32 keyLength)	
Parameters	
cryptoKeyId	Holds the identifier of the key whose key element shall be set.
keyElementId	Holds the identifier of the key element which shall be set.
keyPtr	Holds the pointer to the user-owned key data which shall be set as key element.
keyLength	Contains the length of the key element in bytes.
Return code	
Std_ReturnType	E_OK: Remote service request was dispatched and processed successfully. E_NOT_OK: Due to an internal error, the remote service request could not be dispatched or processed. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match.
Functional Description	
Dispatches remote set key element request to KeyM service. The dispatching functionality shall be only used by a custom Crypto driver on remote side to enable remote service handling.	
Limitations	
The length of the buffer passed as keyPtr must be at least keyLength bytes.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.8.30 KeyM_DispatchRemoteKeyElementGet

Prototype	
Std_ReturnType **KeyM_DispatchRemoteKeyElementGet**(uint32 cryptoKeyId, uint32 keyElementId, uint8* keyPtr, uint32* keyLengthPtr)	
Parameters	
cryptoKeyId	Holds the identifier of the key whose key element shall be retrieved.

continues on next page

Table 4.51 – continued from previous page

keyElementId	Holds the identifier of the key element which shall be retrieved.
keyPtr	Holds the pointer to the user-owned memory location where the key shall be copied to.
keyLengthPtr	Holds a pointer to the memory location in which the output buffer length in bytes is stored. On calling this function, this parameter shall contain the buffer length in bytes of the keyPtr. When the request has finished, the actual size of the written input bytes shall be stored.
Return code	
Std_ReturnType	E_OK: Remote service request was dispatched and processed successfully. E_NOT_OK: Due to an internal error, the remote service request could not be dispatched or processed. KEYM_E_PARAMETER_MISMATCH: Certificate ID invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available, the certificate slot is empty. KEYM_E_KEY_CERT_READ_FAIL: Certificate cannot be provided, access denied.
Functional Description	
Dispatches remote get key element request to KeyM service. The dispatching functionality shall be only used by a custom Crypto driver on remote side to enable remote service handling.	
Limitations	
The length of the buffer passed as keyPtr must be at least *keyLengthPtr bytes.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.31 KeyM_CertElementGetByStructureType

Prototype
Std_ReturnType **KeyM_CertElementGetByStructureType**(const uint8* CertData, uint32 CertDataLength, KeyM_CertificateStructureType CertStructure, uint8* CertElementData, uint32* CertElementDataLength)
Parameters

continues on next page

Table 4.52 – continued from previous page

CertData	Pointer to the certificate data.
CertDataLength	Holds the length of the certificate data.
CertStructure	Holds the certificate structure type.
CertElementData	Pointer to a valid buffer which will hold the data returned by the function.
CertElementDataLength	In: Max number of bytes available in CertElementData. Out: Actual number.
Return code	
Std_ReturnType	E_OK: Element found and data provided in the buffer. E_NOT_OK: Element data not found. KEYM_E_PARAMETER_MISMATCH: Certificate structure type is invalid. KEYM_E_KEY_CERT_SIZE_MISMATCH: Provided buffer for the certificate element too small. KEYM_E_KEY_CERT_EMPTY: No certificate data available.
Functional Description	
Provide certificate element by certificate structure type.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.32 KeyM_CertDelete

Prototype	
Std_ReturnType **KeyM_CertDelete**(KeyM_CertificateIdType CertId)	
Parameters	
CertId	Holds the identifier of the certificate.
Return code	

continues on next page

Table 4.53 – continued from previous page

Std_ReturnType	E_OK: Certificate was deleted successfully. E_NOT_OK: Due to internal error, the certificate could not be deleted. KEYM_E_BUSY: Service cannot be performed yet. Certificate is locked by another service request. KEYM_E_PARAMETER_MISMATCH: The referenced certificate identifier is invalid. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty.
Functional Description	
Deletes certificate from certificate slot. The certificate data is deleted from temporary and/or permanent storage. In addition, the certificate status is reset to KEYM_CERTIFICATE_NOT_AVAILABLE. Besides, valid certificates issued directly or indirectly by the deleted certificates are invalidated to KEYM_CERTIFICATE_PARSED_NOT_VALIDATED. Any certificate information related to the deleted certificate (parse information, public key), is deleted as well.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.2.8.33 KeyM_CertInfoGet

Prototype	
Std_ReturnType **KeyM_CertInfoGet**(KeyM_CertificateIdType CertId, KeyM_CertificateInfoType CertInfoType, uint8* CertInfoData, uint32* CertInfoDataLength)	
Parameters	
CertId	Holds the identifier of the certificate.
CertInfoType	Holds the certificate information type.
CertInfoData	Pointer to a valid buffer which will hold the data returned by the function.
CertInfoDataLength	In: Max number of bytes available in CertInfoData. Out: Actual number.
Return code	

continues on next page

Table 4.54 – continued from previous page

Std_ReturnType	E_OK: Certificate information was retrieved successfully. E_NOT_OK: Internal error or certificate information is not available. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value.
Functional Description	
Retrieve general certificate information.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.34 KeyM_GetCertBasicConstraints

Prototype	
Std_ReturnType **KeyM_GetCertBasicConstraints**(KeyM_CertificateIdType CertId, KeyM_BasicConstraintsType* BasicConstraints)	
Parameters	
CertId	Holds the identifier of the certificate.
BasicConstraints	Basic constraints extension data.
Return code	
Std_ReturnType	E_OK: Certificate extension data was retrieved successfully. E_NOT_OK: Internal error. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty.
Functional Description	
Retrieve basic constraints certificate extension data. The basic constraints extension is specified in RFC5280.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.2.8.35 KeyM_GetCertKeyUsage

Prototype	
Std_ReturnType **KeyM_GetCertKeyUsage**(KeyM_CertificateIdType CertId, KeyM_KeyUsageType* KeyUsage)	
Parameters	
CertId	Holds the identifier of the certificate.
KeyUsage	Key usage extension data.
Return code	
Std_ReturnType	E_OK: Certificate extension data was retrieved successfully. E_NOT_OK: Internal error. KEYM_E_BUSY: Service cannot be performed yet. KeyM is currently busy with other jobs. KEYM_E_PARAMETER_MISMATCH: Parameter does not match with expected value. KEYM_E_KEY_CERT_EMPTY: Certificate slot is empty.
Functional Description	
Retrieve key usage certificate extension data. The key usage extension is specified in RFC5280.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.3 Services used by KeyM

In the following table, services provided by other components which are used by the KeyM are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
CSM	Csm_SignatureVerify
	Csm_KeyElementSet
	Csm_KeySetValid
StbM	StbM_GetCurrentTime
NvM	NvM_ReadBlock

continues on next page

Table 4.57 – continued from previous page

Component	API
	NvM_WriteBlock

Table 4.57 Services used by the KeyM

In order to ensure proper verification, the time reference provided by StbM using the service *StbM_GetCurrentTime* must be trusted.

4.4 Callback Functions

This chapter describes the callback functions that are implemented by the KeyM and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `KeyM_Cbk.h` by the KeyM.

4.4.1 KeyM_CallbackNotificationSignature

Prototype	
void **KeyM_CallbackNotificationSignature**(Crypto_JobType* job, Std_ReturnType result)	
Parameters	
job	Contains the CSM job.
result	Contains the result of the cryptographic operation.
Return code	
void	
Functional Description	
Callback Notification for finished signature verify CSM job. Notifies the KeyM that the signature verify job has finished.	
Limitations	
job has to be a job object handle and a valid pointer. Configuration Variant(s): KEYM_CSM_ASYNC_SIGNATURE_VERIFY == STD_ON	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.4.2 KeyM_NvBlock_ReadFrom_KeyMCertificate_<NvBlock>

Prototype	
Std_ReturnType **KeyM_NvBlock_ReadFrom_KeyMCertificate_<NvBlock>**(const void* NvM-Buffer)	
Parameters	
NvMBuffer	RAM mirror where Ram block data can be read from.
Return code	
Std_ReturnType	E_OK: Data was copied from buffer. E_NOT_OK: Data was not copied.
Functional Description	
Read from Block callback routine. Block specific callback routine which is called by NvM in order to let the KeyM copy data from NvM RAM mirror to RAM block.	
Limitations	
certIdx needs to be valid. NvM storage needs to be preconfigured for referenced certificate.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.4.3 KeyM_NvBlock_WriteTo_KeyMCertificate_<NvBlock>

Prototype	
Std_ReturnType **KeyM_NvBlock_WriteTo_KeyMCertificate_<NvBlock>**(const void* NvMBuffer)	
Parameters	
NvMBuffer	RAM mirror where Ram block data can be read from.
Return code	
Std_ReturnType	E_OK: Data was copied from buffer. E_NOT_OK: Data was not copied.
Functional Description	
Write to Block callback routine. Block specific callback routine which is called by NvM in order to let the KeyM copy data from RAM block to NvM RAM mirror.	
Limitations	

continues on next page

Table 4.60 – continued from previous page

certIdx needs to be valid. NvM storage needs to be preconfigured for referenced certificate. The buffer referenced by NvMBuffer must provide at least KeyM_GetLengthOfNvmBlock(KeyM_GetNvmBlockIdxOfCertificate(certIdx)) byte.
Options
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE

4.4.4 KeyM_NvBlock_Init_KeyMCertificate_<NvBlock>

Prototype	
Std_ReturnType **KeyM_NvBlock_Init_KeyMCertificate_<NvBlock>**( )	
Parameters	
None	
Return code	
Std_ReturnType	E_OK: Data initialized. E_NOT_OK: Any error occurred.
Functional Description	
Init Block callback routine. Block specific callback routine which is called by NvM in order to let the KeyM copy default data to a RAM block.	
Limitations	
certIdx needs to be valid. NvM storage needs to be preconfigured for referenced certificate.	
Options	
Callcontext: TASK Reentrant: FAKSE Synchronous: TRUE	

4.4.5 KeyM_NvBlock_Callback_KeyMCertificate_<NvBlock>

Prototype	
Std_ReturnType **KeyM_NvBlock_Callback_KeyMCertificate_<NvBlock>**(uint8 ServiceId, NvM_RequestResultType JobResult)	
Parameters	
ServiceId	The service identifier of the completed request.
JobResult	Result of the single block job.
Return code	

continues on next page

Table 4.62 – continued from previous page

Std_ReturnType	E_OK: Callback function has been processed successfully. E_NOT_OK: Callback function has not been processed successfully.
Functional Description	
Request finished Block callback routine. Block specific callback routine which is called by NvM in order to notify the KeyM that an asynchronous single block request has been finished.	
Limitations	
certIdx needs to be valid. NvM storage needs to be preconfigured for referenced certificate.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.4.6 KeyM_NvBlock_ReadFrom_CRE

Prototype	
Std_ReturnType **KeyM_NvBlock_ReadFrom_CRE**(const void* NvMBuffer)	
Parameters	
NvMBuffer	RAM mirror where Ram block data can be read from.
Return code	
Std_ReturnType	E_OK: Data was copied from buffer. E_NOT_OK: Data was not copied.
Functional Description	
Block specific callback routine which is called by NvM in order to let the KeyM copy data from NvM RAM mirror to KeyM RAM block.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.4.7 KeyM_NvBlock_WriteTo_CRE

Prototype	
Std_ReturnType **KeyM_NvBlock_WriteTo_CRE**(const void* NvMBuffer)	
Parameters	
NvMBuffer	RAM mirror where Ram block data shall be written to.
Return code	
Std_ReturnType	E_OK: Data was copied to buffer. E_NOT_OK: Data was not copied.
Functional Description	
Block specific callback routine which is called by NvM in order to let the KeyM copy data from RAM block to NvM RAM mirror.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.4.8 KeyM_NvBlock_Init_CRE

Prototype	
Std_ReturnType **KeyM_NvBlock_Init_CRE**( )	
Parameters	
None	
Return code	
Std_ReturnType	E_OK: Data initialized. E_NOT_OK: Any error occurred.
Functional Description	
Block specific callback routine which is called by NvM in order to let the KeyM copy default data to a RAM block.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.4.9 KeyM_NvBlock_Callback_CRE

Prototype	
Std_ReturnType **KeyM_NvBlock_Callback_CRE**(uint8 ServiceId, NvM_RequestResultType JobResult)	
Parameters	
ServiceId	The service identifier of the completed request.
JobResult	Result of the single block job.
Return code	
Std_ReturnType	E_OK: Callback function has been processed successfully. E_NOT_OK: Callback function has not been processed successfully.
Functional Description	
Block specific callback routine which is called by NvM in order to notify the KeyM that an asynchronous single block request has been finished.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.5 Configurable Interfaces

4.5.1 Notifications

At its configurable interfaces the KeyM defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the KeyM but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

4.5.1.1 Appl_VerifyCallbackFunc

Prototype	
Std_ReturnType **<Appl_VerifyCallbackFunc>**(KeyM_CertificateIdType CertId, KeyM_CertificateStatusType Result)	
Parameters	
CertId	The certificate identifier that has been verified.
Result	Contains information about the result of the operation.
Return code	

continues on next page

Table 4.67 – continued from previous page

Std_ReturnType	E_OK: Operation successful. E_NOT_OK: Operation failed.
Functional Description	
Indicate result of the verification operation. Notifies the application that a certificate verification has been finished. The function name is configurable by KeyMCertificateVerifyCallbackNotificationFunc.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.5.1.2 Appl_ServiceCallbackFunc

Prototype	
void **<Appl_ServiceCallbackFunc>**(KeyM_CertificateIdType CertId, KeyM_ResultType Result, uint16 ResultDataLength, uint8* ResultDataPtr)	
Parameters	
CertId	Certificate identifier.
Result	Contains information about the result of the operation.
ResultDataLength	Contains the length of the resulting data of this operation if any.
ResultDataPtr	Pointer to the data of the result. Is only guaranteed to be valid if ResultDataLength > 0.
Return code	
void	
Functional Description	
Indicate the end of a service operation. Notifies the application that the certificate service operation has been finished. This function is used by the certificate submodule. This callback is only provided if KeyMServiceCertificateFunctionEnabled is set to TRUE. The function name is configurable by KeyMServiceCertificateCallbackNotificationFunc.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.5.1.3 Appl_VerifyGroupCallbackFunc

Prototype	
Std_ReturnType **Appl_VerifyGroupCallbackFunc**(KeyM_CertificateGroupIdType GroupId, KeyM_CertificateGroupStatusType Result)	
Parameters	
GroupId	Holds the certificate group identifier.
Result	Contains information about the result of the operation.
Return code	
Std_ReturnType	E_OK: Operation successful. E_NOT_OK: Operation failed.
Functional Description	
Indicate result of the certificate group verification operation. Notifies the application that a certificate group verification has been finished. The function name is configurable by KeyMCertificateGroupVerifyCallbackNotificationFunc.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: FALSE	

4.5.2 Callout Functions

At its configurable interfaces the KeyM defines callout functions. The declarations of the callout functions are provided by the BSW module, i.e. the KeyM. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The KeyM callout function declarations are described in the following tables:

4.5.2.1 Appl_CertificateElementVerificationCallout

Prototype	
Std_ReturnType **Appl_CertificateElementVerificationCallout**(KeyM_CertificateIdType CertId, KeyM_CertElementIdType CertElementId, const uint8* CertElementData, uint32 CertElement-DataLength)	
Parameters	
CertId	Certificate identifier.
CertElementId	Certificate element identifier.
CertElementData	Pointer to certificate element data.
CertElementDataLength	Length of certificate element data.
Return code	

continues on next page

Table 4.70 – continued from previous page

Std_ReturnType	E_OK: Element verification successful. E_NOT_OK: Element verification failed.
Functional Description	
Verify certificate elements. Callout to verify that a given certificate fulfills the specified rules and conditions.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.5.2.2 Appl_SetKeyCallout

Prototype	
Std_ReturnType **Appl_SetKeyCallout**(KeyM_CertificateIdType CertId, KeyM_CertElementIdType CertElementId, uint32 csmKeyId, const uint8* CertElementData, uint32 CertElementDataLength)	
Parameters	
CertId	Certificate identifier.
CertElementId	Certificate element identifier.
csmKeyId	CSM key identifier.
CertElementData	Pointer to certificate element data.
CertElementDataLength	Length of certificate element data.
Return code	
Std_ReturnType	E_OK: Operation successful. E_NOT_OK: Operation failed.
Functional Description	
Set Key Callout. Callout to set the certificate key if the format is not supported by the KeyM.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.5.2.3 Appl_CertInitCallout

Prototype	
void **Appl_CertInitCallout**(KeyM_CertificateIdType CertId, KeyM_ConstCertDataType* CertificateDataPtr)	
Parameters	

continues on next page

Table 4.72 – continued from previous page

CertId	Certificate identifier.
CertificateDataPtr	Provides a pointer to a certificate data structure. The buffer located by the pointer in the structure should be provided by the caller of this function. The length information indicates the maximum length of the buffer when the function is called. When the function returns, the length information indicates the actual length of the certificate data in the buffer.
Return code	
void	
Functional Description	
Provide the certificate data during initialization.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.5.2.4 Appl_GetCurrentTimeCalloutFunc

Prototype	
Std_ReturnType **Appl_GetCurrentTimeCalloutFunc**(KeyM_CertificateIdType CertId, uint64* timeStamp)	
Parameters	
CertId	Certificate identifier.
timeStamp	Current time in Unix time format.
Return code	
Std_ReturnType	E_OK: Operation successful. E_NOT_OK: Operation failed. KEYM_E_NO_PERIOD_VALIDITY_CHECK: Skip time stamp validation.
Functional Description	
Provide current time to verify certificate time stamp.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

5 Configuration

The functionalities of the module are configured with DaVinci Configurator Classic.

For details on configuration options refer to the module-specific description that is shown in DaVinci Configurator Classic.

5.1 Configuration Variants

This module supports these configuration variants:

> VARIANT-PRE-COMPILE

The configuration classes of the parameters depend on the supported configuration variants. For their definitions please see the module-specific BSWMD ARXML file.

5.2 Certificate Elements

Please make sure that following configuration requirements for certificate elements are met:

Caution
The object type (`KeyMCertificateElementObjectType`) of a certificate element needs to be configured. Please note that only the types listed in chapter *Object Type* are currently supported.

Caution
The structure type (`KeyMCertificateElementOfStructure`) of a certificate element needs to be configured according to the contained element types depending on the used certificate format. For more details, please refer to the standards of X.509, CRL *[7]* and CVC *[12]*.

Caution
If a certificate element is contained in a structure with a given object identifier, this OID needs to be configured in `KeyMCertificateElementObjectId`.

5.3 NvM Block Needs

The following table includes the configuration constraint for the used NvM blocks.

Parameter	Block Type	Expected Value
NvMBlockUseSetRamBlockStatus	–	TRUE
NvMBlockUseSyncMechanism	–	TRUE
NvMInitBlockCallback	–	Init Block Callback
NvMReadRamBlockFromNvCallback	–	Read from Block Callback
NvMSelectBlockForReadAll	–	TRUE
NvMSelectBlockForWriteAll	DEFERRED	TRUE
NvMSelectBlockForWriteAll	IMMEDIATE	FALSE
NvMSetRamBlockStatusApi	–	TRUE
NvMSingleBlockCallback	–	Block Callback
NvMWriteRamBlockToNvCallback	–	Write to Block Callback
MICROSAR/NvMInvokeCallbacksForWriteAll	–	TRUE
MICROSAR/NvMUseInitCallback	–	TRUE
MICROSAR/NvMUseJobendCallback	–	TRUE

Table 5.1 NvM Block Needs

5.4 FAQ

What are possible causes for the failure of certificate parsing?

- Parsing error is caused if the passed certificate data does not correspond to the certificate configuration. Please check the certificate identifier.
- Parsing error is caused if the `KeyMCertAllowUnconfiguredElements` parameter is disabled and not all certificate elements are configured.
- Parsing error is caused if the `KeyMCertAllowFlexibleOrder` parameter is disabled and the certificate elements are not configured in the exact order as they appear in the certificate data.
- Parsing error is caused if the configured object type for a certificate element is not supported by the KeyM (`KeyMCertificateElementObjectType`) or is not a primitive ASN.1 tag identifier.
- Parsing error is caused if the configured structure type for a certificate element does not correspond to the specified data structures for X.509, CRL and CVC (`KeyMCertificateElementOfStructure`).
- Parsing error is caused if the configured object identifier does not correspond to the object identifier contained in the certificate data (`KeyMCertificateElementObjectId`).
- Parsing error is caused if the configured element path for the certificate sub-element is not configured correctly (`KeyMCertificateElement`). Please refer to *Element configuration*.

What are possible causes for the failure of certificate verification?

- Verification error is caused if the CSM jobs for the signature verification and the corresponding CSM keys for the public keys are not configured correctly. Please refer to *Verification Job and Key Dependencies*.
- Verification error is caused if the configured upper hierarchical reference for a certificate (`KeyMCertUpperHierarchicalCertRef`) does not correspond to the actual issuer of the certificate.
- Verification error is caused if no separate CSM jobs and CSM keys are used for each individual certificate.
- Verification error is caused if the parsed validity period in the certificate data is not valid.
- Verification error is caused if the CSM operation for setting the public key or verifying the certificate signature fails due to internal error.

6 Cybersecurity

This chapter describes relevant information for a secure integration and configuration, so-called Cybersecurity Manual Instructions (CMI), of this component to fulfil identified Technical Cybersecurity Requirements (TCR). Additionally, functional security dependencies to other components are described.

6.1 Integration

6.1.1 CMI-KeyM-Callouts

Technical Cybersecurity Requirements: TCR-MSRC-AccessCertificates, TCR-MSRC- HandleCertificates

The user of MICROSAR Classic shall ensure when using KeyM callouts, that the result of the callout is as expected.

Rationale: Through callouts, the user can enhance KeyM functionality e.g., verify certificate elements, store keys, and provide the current time. If any of these user-implemented callouts does not provide the expected result it could lead to security vulnerabilities in the system.

6.2 Configuration

6.2.1 CMI-KeyM-CertificateChainDepth

Technical Cybersecurity Requirements: TCR-MSRC-AccessCertificates, TCR-MSRC- HandleCertificates

The user of MICROSAR Classic shall ensure that the certificate chain with the most certificates is still within the limits of *Certificate Chain Max Depth*.

Rationale: The configuration parameter *CertificateChainMaxDepth* has a validation that only checks the statically configured certificate chains. If a dynamic certificate chain exceeds this number, it can lead to certificates retaining the “validated” state even though a certificate higher up in the chain has been revoked.

6.2.2 CMI-KeyM-NvWriteBlockFctName

Technical Cybersecurity Requirements: TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall ensure that the function configured for *Nv Write Block Fct Name* is the correct function for writing NvM blocks.

Rationale: If the function given in this parameter does not write NvM blocks as expected, it could lead to CRE entries not being persisted, which in turn could lead to certificates being considered valid that were already revoked.

6.2.3 CMI-KeyM-CREEntryMaxLength

Technical Cybersecurity Requirements: TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall ensure that the values of *CRE Max Length Of Issuer Name* and *CRE Max Length Of Serial Number* is sufficient to hold all certificate data that shall be stored.

Rationale: If these values are selected too small, it could lead to CRE entries not being persisted, which in turn could lead to certificates being considered valid that were already revoked. (However, the function for setting a CRE will return E_NOT_OK if this happens.)

6.2.4 CMI-KeyM-CREMaxNumberOfEntries

Technical Cybersecurity Requirements: TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall ensure that the value of *CRE Max Number Of Entries* is sufficient to hold all certificate entries that shall be stored over the lifetime of the system.

Rationale: CRE entries are persisted for the lifetime of the system. It is not possible to overwrite existing CRE entries if the maximum number of entries is reached. Therefore, the value of *CRE Max Number Of Entries* shall have enough safety margin to not run out of space.

6.2.5 CMI-KeyM-CertificateSlotSharing

Technical Cybersecurity Requirements: TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall ensure that multiple certificates that share a slot can never be installed at the same time.

Rationale: If a certificate slot is already occupied and another certificate is received that shares the same slot, the saved certificate will be overwritten.

6.2.6 CMI-KeyM-PublishedInformation

Technical Cybersecurity Requirements: TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall ensure when enabling *Remote CRE Enabled* and/or *Remote OCSP Enabled* that the functions for forwarding the CRE/OCSP information to HSM are working as expected.

Rationale: If these functions do not forward the information correctly to HSM, it could lead to CRE entries not being persisted, which in turn could lead to certificates being considered valid that were already revoked.

6.2.7 CMI-KeyM-NvBlockProcessing

Technical Cybersecurity Requirements: TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall set *CRE Nv Block Processing* and/or *Cert Nv Block Processing* to DEFERRED to prevent inconsistencies when updating CREs.

Rationale: If the parameter is set to IMMEDIATE and errors occur while updating the status of a certificate chain, it could happen that CREs for some certificates are not set or that certificate data is not updated correctly.

6.2.8 CMI-KeyM-NvBlockReference

Technical Cybersecurity Requirements: TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall ensure that *Nvm Block Descriptor Ref* references a valid NvM block.

Rationale: If the reference is not correct, the KeyM cannot load the certificate on startup and a verification of certificates that are below in the chain will fail.

6.3 Runtime Interfaces: Application

6.3.1 CMI-KeyM-Initialization

Technical Cybersecurity Requirements: TCR-MSRC-HandleCertificates

The user of MICROSAR Classic shall ensure that the `KeyM_InitMemory()` and `KeyM_Deinit()` APIs are not called while KeyM functionality is still required.

Rationale: The above-mentioned APIs change the initialization state of the KeyM to “KEYM_UNINIT and could therefore lead to unavailability of KeyM services if called during runtime.

6.3.2 CMI-KeyM-CertificateDeletion

Technical Cybersecurity Requirements: TCR-MSRC-AccessCertificates

The user of MICROSAR Classic shall ensure that the `KeyM_CertDelete()` API is only called for certificates that should really be deleted.

Rationale: It is highly recommended not to use this API at all, because the deletion of a certificate not only affects the certificate itself, but also the validation status of certificates that are lower in the chain. If it is still necessary to delete a certificate, countermeasures should be taken to avoid deleting the wrong certificate.

6.4 Runtime Interfaces: BSW

TCR	Depends on Components(s)	Comment
TCR-MSRC-AccessCertificates	CSM, NvM	–
TCR-MSRC-HandleCertificates	CSM	–

Table 6.1 Cybersecurity-relevant Dependencies for Runtime Interfaces

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
BswM	BSW Mode Manager
CSM	Cryptographic Service Manager
NvM	NVRAM Manager
SchM	BSW Scheduler Module
StbM	Synchronized Time-Base Manager

Table 7.1 Glossary

7.2 Abbreviations

Abbreviation	Description
AC	Attribute Certificate
API	Application Programming Interface
ASN	Abstract Syntax Notation
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CA	Certificate Authority
CRE	Certificate Revocation Entry
CRL	Certificate Revocation List
CSR	Certificate Signing Request
DER	Distinguished Encoding Rules
DET	Development Error Tracer
ECC	Elliptic Curve Cryptography
ECU	Electronic Control Unit
HSM	Hardware Security Module
OCSP	Online Certificate Status Protocol
PKI	Public Key Infrastructure
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7.2 Abbreviations

8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com