Document Title	Specification of Key Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	907

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

Document Change History

Date	Release	Changed by	Change Description
2021-11-25	R21-11	AUTOSAR Release Management	• Editorial changes. • Add upstream requirements
2020-11-30	R20-11	AUTOSAR Release Management	• Editorial changes, improve error section • Add security events for IdsM • Detail order of certificate verification • Align functions, parameters and return values for C-API and service interfaces • Signing request reference for CSR
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes. • Create general error detection in chapter 7.4. • Changed Document Status from Final to published.
2018-10-31	4.4.0	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

The AUTOSAR KeyM module consists of two sub modules, the crypto key submodule and the certificate submodule.

The crypto key submodule provides an API and configuration items to introduce or update pre-defined cryptographic key material. It acts as a key client to interpret the provided data from a key server and to create respective key materials. These keys are provided to the crypto service manager. After successful installation of the key material, the application is able to utilize the crypto operations. This allows OEMs to introduce key materials in production or maintenance phase to ECUs separate from the application.

The certificate submodule provides an API and configuration to operate on certificates. It allows to define certificate slots and associate them in a hierarchy as it is used in a PKI. Certificates can be permanently stored like a Root or intermediate certificate(s) so that they can be used to verify a given certificate against a certificate chain. Furthermore, the certificate submodule allows to access certificate elements or to verify its contents.

1.1 Important note

This specification provides skeletons of an API for a Vehicle Key and Certificate Management system. Not all functionalities have been completely specified. This may allow some freedom of interpretation and implementation details. Even though the interfaces have been designed in a generic and flexible way it might be the case that they can change in upcoming AUTOSAR releases.

2 Acronyms and abbreviations

Abbreviation / Acronym:	***Description:***
KeyM	Key Manager
PKI	Public Key Infrastructure
CSR	Certificate Signing Request
CSM	Crypto Service Manager
CRL	Certificate Revocation List
CA	Certificate Authority
OID	Object Identifier. A byte array that identifies a certificate element or group or list of certificate elements.

3 Related documentation

3.1 Input documents

[1] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] AUTOSAR General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] AUTOSAR General Specification for Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

[4] AUTOSAR Specification of Crypto Service Manager
AUTOSAR_SWS_CryptoServiceManager.pdf

[5] AUTOSAR Requirements on Crypto Stack
AUTOSAR_SRS_CryptoStack.pdf

[6] AUTOSAR Requirements on Intrusion Detection System
AUTOSAR_RS_IntrusionDetectionSystem.pdf

3.2 Related standards and norms

[7] IEC 7498-1 The Basic Model, IEC Norm, 1994

[8] IETF 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

[9] SHE – Secure Hardware Extension, Functional Specification, V1.1

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software (SWS BSW General) [3] which is also valid for the Key Management module.

Thus, the specification SWS BSW General [3] shall be considered as additional and required specification for the Key and Certificate Management module.

4 Constraints and assumptions

4.1 Limitations

The Key Management module shall be used with a Crypto Service Manager and its underlying modules.

Only a single KeyElement (with ID = 1) per CsmKey is currently supported.

4.2 Applicability to car domains

This specification has no limitations to specific car domains.

5 Dependencies to other modules

This chapter lists the relations to other modules that are used by the AUTOSAR KeyM module.

5.1 Dependencies to Crypto Service Manager

The KeyM module depends on cryptographic algorithms and functions provided by the Csm module. The KeyM module requires API functions to retrieve and set key elements and to verify signatures of certificates, namely:

- Key Setting Interface
- Key Extraction Interface
- Key Copying Interface
- Key Generation Interface
- Key Derivation Interface
- Key Exchange Interface
- Signature Interface

5.2 Dependencies to Non Volatile Memory

The KeyM can be configured to store key material in non volatile memory. This requires interfaces to NVM.

5.3 Dependencies to Synchronized Time Base

The time for certificate validation period is provided by the STBM.

6 Requirements traceability

Requirement	Description	Satisfied by
RS_Ids_00810	Basic SW security events	SWS_KeyM_00171, SWS_KeyM_00172, SWS_KeyM_00173
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_KeyM_00043
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_KeyM_00043
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_KeyM_00157
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_KeyM_00049
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_KeyM_00043, SWS_KeyM_00158
SRS_CryptoStack_00003	The crypto stack shall be able to incorporate modules of the crypto library	SWS_KeyM_00174
SRS_CryptoStack_00006	Each primitive of the CRYIF shall belong to exactly one service of the CSM	SWS_KeyM_00174
SRS_CryptoStack_00007	The Crypto Stack shall provide scalability for the cryptographic features	SWS_KeyM_00001, SWS_KeyM_00002
SRS_CryptoStack_00008	The Crypto Stack shall allow static configuration of keys used for cryptographic jobs	SWS_KeyM_00174
SRS_CryptoStack_00009	The Crypto Stack shall support reentrancy for all crypto services	SWS_KeyM_00174
SRS_CryptoStack_00010	The Crypto Stack shall conceal symmetric keys from the users of crypto services	SWS_KeyM_00091
SRS_CryptoStack_00013	The modules of the crypto stack shall support only pre-compile time configuration	SWS_KeyM_00001, SWS_KeyM_00002, SWS_KeyM_00007, SWS_KeyM_00010
SRS_CryptoStack_00014	The Crypto Interface shall have an interface to the static configuration information of the Crypto Driver	SWS_KeyM_00174
SRS_CryptoStack_00015	Channels mapped to different Crypto Driver Objects shall be	SWS_KeyM_00174

	uniquely configurable in Crypto Interface	
SRS_CryptoStack_00022	The Crypto Stack shall identify MAC generation/verification as a cryptographic primitive which can be requested to a driver	SWS_KeyM_00108
SRS_CryptoStack_00023	The Crypto Stack shall identify asymmetric signature generation/verification as a cryptographic primitive which can be requested to a driver	SWS_KeyM_00113, SWS_KeyM_00136
SRS_CryptoStack_00027	The Crypto Stack shall provide an interface for the generation of symmetric keys	SWS_KeyM_00089, SWS_KeyM_00091, SWS_KeyM_00100
SRS_CryptoStack_00028	The Crypto Stack shall provide an interface for key exchange mechanisms	SWS_KeyM_00003, SWS_KeyM_00004, SWS_KeyM_00005, SWS_KeyM_00085, SWS_KeyM_00086
SRS_CryptoStack_00031	The Crypto Stack shall provide an interface for parsing certificates	SWS_KeyM_00045, SWS_KeyM_00134, SWS_KeyM_00135, SWS_KeyM_00139
SRS_CryptoStack_00034	The Crypto Interface shall report detected development errors to the Default Error Tracer	SWS_KeyM_00174
SRS_CryptoStack_00036	The Crypto Driver shall allow static configuration of Crypto Driver Objects	SWS_KeyM_00174
SRS_CryptoStack_00061	The Crypto Stack shall support detection of invalid keys	SWS_KeyM_00113
SRS_CryptoStack_00075	The Crypto Interface shall be the interface layer between the underlying crypto driver(s) and upper layers	SWS_KeyM_00174
SRS_CryptoStack_00076	The Crypto Interface implementation and interface shall be independent from underlying Crypto Hardware or Software	SWS_KeyM_00174
SRS_CryptoStack_00079	The job processing mode (synchronous or asynchronous) of a CSM service shall be defined by static configuration	SWS_KeyM_00174
SRS_CryptoStack_00080	The set of cryptographic services provided by the CSM shall be defined by static configuration	SWS_KeyM_00021
SRS_CryptoStack_00081	The CSM module specification shall specify which other modules are required	SWS_KeyM_00174

SRS_CryptoStack_00082	The CSM module specification shall specify the interface and behavior of the callback function, if the asynchronous job processing mode is selected	SWS_KeyM_00174
SRS_CryptoStack_00084	The CSM module shall use the streaming approach for some selected services	SWS_KeyM_00174
SRS_CryptoStack_00086	The CSM module shall distinguish between error types	SWS_KeyM_00155
SRS_CryptoStack_00087	The CSM module shall report detected development errors to the Default Error Tracer	SWS_KeyM_00044, SWS_KeyM_00144, SWS_KeyM_00145, SWS_KeyM_00146
SRS_CryptoStack_00088	The CSM module shall provide an abstraction layer which offers a standardized interface to higher software layers to access cryptographic algorithms	SWS_KeyM_00174
SRS_CryptoStack_00089	The CSM module shall be located in the AUTOSAR service layer	SWS_KeyM_00174
SRS_CryptoStack_00090	The CSM shall provide an interface to be accessible via the RTE	SWS_KeyM_00160, SWS_KeyM_00161, SWS_KeyM_00162, SWS_KeyM_00163, SWS_KeyM_00164
SRS_CryptoStack_00091	The CSM shall provide one Provide--Port for each configuration	SWS_KeyM_00160, SWS_KeyM_00161, SWS_KeyM_00162, SWS_KeyM_00163, SWS_KeyM_00164
SRS_CryptoStack_00095	The Crypto Driver module shall strictly separate error and status information	SWS_KeyM_00174
SRS_CryptoStack_00096	The CSM module shall not return specific development error codes via the API	SWS_KeyM_00009, SWS_KeyM_00085, SWS_KeyM_00086, SWS_KeyM_00090, SWS_KeyM_00099, SWS_KeyM_00104, SWS_KeyM_00116, SWS_KeyM_00117, SWS_KeyM_00119, SWS_KeyM_00121, SWS_KeyM_00125, SWS_KeyM_00128, SWS_KeyM_00132, SWS_KeyM_00141, SWS_KeyM_00155, SWS_KeyM_00166, SWS_KeyM_00174
SRS_CryptoStack_00097	The CSM shall check passed API parameters for validity	SWS_KeyM_00174
SRS_CryptoStack_00098	The Crypto Driver shall provide access to all cryptographic algorithms supported by the hardware	SWS_KeyM_00174
SRS_CryptoStack_00101	Asynchronous Job Processing	SWS_KeyM_00094, SWS_KeyM_00109, SWS_KeyM_00120, SWS_KeyM_00124, SWS_KeyM_00149, SWS_KeyM_00151

SRS_CryptoStack_00102	The priority of a user and its crypto jobs shall be defined by static configuration	SWS_KeyM_00174
SRS_CryptoStack_00103	The Crypto Stack shall provide an interface for the derivation of symmetric keys	SWS_KeyM_00003
SRS_CryptoStack_00104	Crypto Interface keys mapped to different Crypto Driver Keys shall be uniquely configurable in the Crypto Interface	SWS_KeyM_00174
SRS_CryptoStack_00105	The Crypto Stack shall only allow unique key identifiers	SWS_KeyM_00013, SWS_KeyM_00091
SRS_CryptoStack_00106	Key manager operation shall either run synchronously or asynchronously.	SWS_KeyM_00080, SWS_KeyM_00095, SWS_KeyM_00105, SWS_KeyM_00109, SWS_KeyM_00119, SWS_KeyM_00120, SWS_KeyM_00124, SWS_KeyM_00149, SWS_KeyM_00150, SWS_KeyM_00151, SWS_KeyM_00152, SWS_KeyM_00153, SWS_KeyM_00156
SRS_CryptoStack_00107	Key manager shall provide interfaces to generate or update key material.	SWS_KeyM_00012, SWS_KeyM_00087, SWS_KeyM_00089, SWS_KeyM_00092, SWS_KeyM_00093, SWS_KeyM_00095, SWS_KeyM_00096, SWS_KeyM_00097, SWS_KeyM_00099, SWS_KeyM_00100, SWS_KeyM_00101, SWS_KeyM_00102, SWS_KeyM_00103, SWS_KeyM_00104, SWS_KeyM_00156
SRS_CryptoStack_00108	Key manager shall be able to negotiate a shared secret by exchanging messages with other ECUs	SWS_KeyM_00011
SRS_CryptoStack_00109	Key manager shall be able to manage derivation of key material from a common secret	SWS_KeyM_00089, SWS_KeyM_00096
SRS_CryptoStack_00110	The KeyM module shall support on-board generated keys	SWS_KeyM_00011
SRS_CryptoStack_00111	The KeyM module shall support verification of certificates based on configured rules	SWS_KeyM_00022, SWS_KeyM_00024, SWS_KeyM_00027, SWS_KeyM_00028, SWS_KeyM_00029, SWS_KeyM_00030, SWS_KeyM_00031, SWS_KeyM_00032, SWS_KeyM_00033, SWS_KeyM_00034, SWS_KeyM_00035, SWS_KeyM_00045, SWS_KeyM_00110, SWS_KeyM_00111, SWS_KeyM_00112, SWS_KeyM_00113, SWS_KeyM_00114, SWS_KeyM_00115, SWS_KeyM_00118, SWS_KeyM_00135, SWS_KeyM_00139, SWS_KeyM_00168, SWS_KeyM_00169
SRS_CryptoStack_00112	The KeyM module shall support retrieving arbitrary elements of a certificate	SWS_KeyM_00117, SWS_KeyM_00127, SWS_KeyM_00128, SWS_KeyM_00129, SWS_KeyM_00130, SWS_KeyM_00131, SWS_KeyM_00132, SWS_KeyM_00148

SRS_CryptoStack_00113	Keys in the crypto stack can be uniquely identified	SWS_KeyM_00013, SWS_KeyM_00091
SRS_CryptoStack_00114	Crypto driver shall place keys into specific key slots	SWS_KeyM_00046, SWS_KeyM_00154
SRS_CryptoStack_00115	KeyM shall be highly configurable to support different OEM use cases	SWS_KeyM_00006, SWS_KeyM_00011, SWS_KeyM_00088, SWS_KeyM_00089, SWS_KeyM_00133, SWS_KeyM_00134, SWS_KeyM_00136, SWS_KeyM_00137, SWS_KeyM_00138, SWS_KeyM_00140, SWS_KeyM_00141, SWS_KeyM_00167
SRS_CryptoStack_00117	Keys shall not be used if they are empty or corrupted	SWS_KeyM_00023, SWS_KeyM_00026, SWS_KeyM_00098
SRS_CryptoStack_00118	Key material shall be securely stored either in NVM or CSM	SWS_KeyM_00008, SWS_KeyM_00011, SWS_KeyM_00014, SWS_KeyM_00016, SWS_KeyM_00017, SWS_KeyM_00018, SWS_KeyM_00019, SWS_KeyM_00022, SWS_KeyM_00023, SWS_KeyM_00046, SWS_KeyM_00098, SWS_KeyM_00123, SWS_KeyM_00126, SWS_KeyM_00166, SWS_KeyM_00170
SRS_CryptoStack_00119	Provide a proof that the key has been programmed correctly	SWS_KeyM_00009, SWS_KeyM_00015, SWS_KeyM_00019, SWS_KeyM_00020, SWS_KeyM_00107, SWS_KeyM_00108
SRS_CryptoStack_00120	Cleanup all key material on shutdown operation	SWS_KeyM_00025, SWS_KeyM_00048, SWS_KeyM_00106
SRS_CryptoStack_00121	Pass security related events to security event memory (SEM) for secure logging	SWS_KeyM_00171, SWS_KeyM_00173
SRS_CryptoStack_00122	Log security events reported by basic software modules and SWC	SWS_KeyM_00174
SRS_CryptoStack_00123	Configure security event properties	SWS_KeyM_00174
SRS_CryptoStack_00124	Allow authorized users to read SEM data via diagnostic interfaces	SWS_KeyM_00174

7 Functional specification

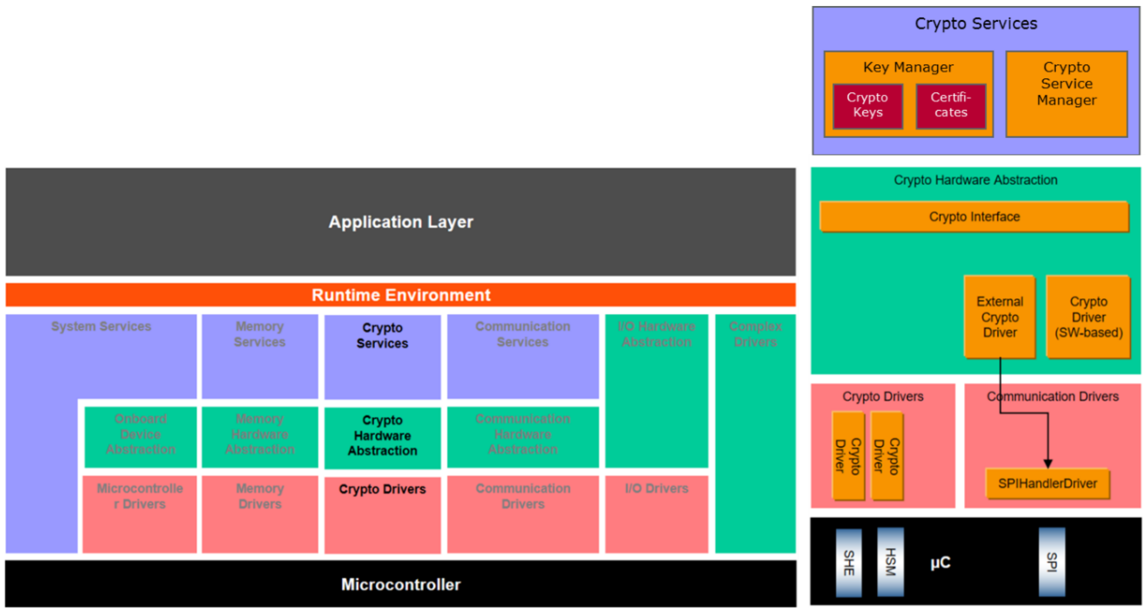


Figure 7-1: AUTOSAR layered view with KEYM

The Key Management module can roughly be divided into two parts: the crypto key sub module and the certificate sub module. The crypto key sub module is mainly used to interact with a key provisioning entity (key master) that initiates the generation or provides key material directly. These keys are assigned to crypto keys of the CSM and stored in dedicated NVM blocks or can be stored as keys of the respective crypto driver. The certificate sub module allows to configure certificates of a chain, providing interfaces to store and verify them. The public key contained in a certificate can further be assigned to CSM keys so that they can be used by crypto jobs.

[SWS_KeyM_00001] ⌈ The crypto key sub module of the Key Manager shall be completely disabled if *KeyMCryptoKeyManagerEnabled* is set to FALSE. No function shall be available, and no resources shall be allocated in this case that is not needed for other operation.
⌋(SRS_CryptoStack_00013, SRS_CryptoStack_00007)

[SWS_KeyM_00002] ⌈ The support of the certificate sub module within the Key Manager shall be completely disabled if *KeyMCertificateManagerEnabled* is set to FALSE. No function shall be available and no resources shall be allocated in this case that is associated to certificate operations.
⌋(SRS_CryptoStack_00013, SRS_CryptoStack_00007)

7.1 Crypto key submodule

The crypto key submodule is used to initialize, update and maintain cryptographic key material for an ECU. One use case is the provision of keys for the secured on-

board communication that need to be distributed to the involved ECUs. These keys should be provided to CSM keys which are assigned to crypto jobs that are used for authentication of Secured I-PDUs. It is therefore crucial from a modelling aspect to assign the keys provided by the key master to the CSM keys and jobs used for the respective Secured-I PDUs. This is an overall task in a vehicle and affects several ECUs in the same way. It is one purpose of the crypto key submodule to support this operation.
The key master can either be located directly in the vehicle to coordinate the key generation internally, e.g. as a particular ECU. It is also possible to use a backend system in the cloud that generates the key material and provides the necessary data in a secure way to the ECUs. Usually diagnostic commands are used for the communication, directly or indirectly, between the key master and the crypto key sub module.

7.1.1 General behavior

[SWS_KeyM_00003] ⌈ The crypto key submodule can be configured to perform crypto key operation in a session like manner. In this way, key operation such as `KeyM_Prepare()` or `KeyM_Update()` are only accepted during an open session.
⌋(SRS_CryptoStack_00028, SRS_CryptoStack_00103)

[SWS_KeyM_00004] ⌈ A session is started by a call to `KeyM_Start()`. Afterwards key operations can be performed until the session is closed with a call to the function `KeyM_Finalize()`.
⌋(SRS_CryptoStack_00028)

[SWS_KeyM_00005] ⌈ By default, the `KeyM_Start()` function will not consider any input data or length information and will not provide any output data nor will the output data length be changed.
⌋(SRS_CryptoStack_00028)

[SWS_KeyM_00006] ⌈ Optionally, a key handler can be called if the configuration option *KeyMCryptoKeyHandlerStartFinalizeEnabled* is set to `TRUE`. The `KeyM_Start()` function will call in turn the `KeyM_KH_Start()` function with the same parameter of `KeyM_Start()`. The return value of `KeyM_KH_Start()` will be used as the return value of `KeyM_Start()`.
⌋(SRS_CryptoStack_00115)

<u>Rationale:</u>

The `KeyM_KH_Start()` function can perform OEM specific checks like signature verification of any input data to prove the authenticity for a key management operation.

Note: The definition of *KeyMCryptoKeyHandlerStartFinalizeEnabled* has only effect if *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to `TRUE`.

[SWS_KeyM_00007] ⌈ If the configuration option *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to `FALSE`, the function `KeyM_Start()` and `KeyM_Finalize()` are not provided by the Key Management module. A key update operation can then be performed at any time.
⌋(SRS_CryptoStack_00013)

[SWS_KeyM_00008] ⌈ A session is closed by a call to `KeyM_Finalize()`. During the call, all keys that were updated within the session will be set to valid by calling `Csm_KeySetValid()`. After the function has been completed its operation, no further key update operations will be accepted.
⌋(SRS_CryptoStack_00118)

[SWS_KeyM_00009] ⌈ The function `KeyM_Finalize()` will return `E_OK` if all keys have been validated successfully. If at least one key could not be validated successfully, the function shall return `E_NOT_OK`. Nevertheless, all keys shall be validated that have been updated and the operation shall not be aborted if one key validation has failed.
⌋(SRS_CryptoStack_00119, SRS_CryptoStack_00096)

[SWS_KeyM_00010] ⌈ If the configuration option *KeyMCryptoKeyPrepareFunctionEnabled* is set to `TRUE` the function `KeyM_Prepare()` is provided. This function has currently no functional behavior. If the configuration option is set to `FALSE`, the functional interface is not provided.
⌋(SRS_CryptoStack_00013)

[SWS_KeyM_00011] ⌈ If the configuration option *KeyMCryptoKeyHandlerPrepareEnabled* is set to `TRUE`, then a call to `KeyM_Prepare()` will in turn passed on to `KeyM_KH_Prepare()` and the arguments and return value will be passed accordingly.
⌋(SRS_CryptoStack_00108, SRS_CryptoStack_00115, SRS_CryptoStack_00118, SRS_CryptoStack_00110)

<u>Rationale:</u>

The intention is to call `KeyM_Prepare()` once at the beginning after the key update session has been initiated. The calling diagnostic service can provide specific data to the key handler which is needed to perform the following key update operation. For example, it could be used to extract crypto driver specific information needed by the key master which is extracted from the (SHE-)hardware and provided in the output buffer back again. Or it can initiate an OEM specific key negotiation process with results that are later on necessary for the key update process. Another possibility would be, that a (encrypted) common key is provided by the key master during preparation. The specific key handler is able to (decrypt and) store the key in the

CSM. This results in a common key that is assigned to a CSM key and can further be used to derive other keys from it.

[SWS_KeyM_00012] ⌈ A key update is triggered by a call to `KeyM_Update()`, typically initiated by a diagnostic service.
⌋(SRS_CryptoStack_00107)

[SWS_KeyM_00013] ⌈ If `KeyM_Update()` is called and *KeyMCryptoKeyHandlerUpdateEnabled* is set to `FALSE` and keyNameLength is greater than 0, the crypto key submodule will search for the key name configured in KeyMCryptoKey/*KeyMCryptoKeyName*. If the key name is not found, the function will return `E_NOT_OK`. If found, the function will trigger the key update operation.
⌋(SRS_CryptoStack_00113, SRS_CryptoStack_00105)

[SWS_KeyM_00014] ⌈ If `KeyM_Update()` is called and *KeyMCryptoKeyHandlerUpdateEnabled* is set to FALSE and keyNameLength is 0, the crypto key submodule will interpret the input data as M1M2M3 values of a SHE key. The key_ID is extracted from M1 by extracting bit 121..124 of the input data and will search for the corresponding value in KeyMCryptoKeyCryptoProps to identify the *KeyMCryptoKeyId* and the associated CsmKeyRef. If found, the function will trigger the keyupdate operation.
⌋(SRS_CryptoStack_00118)

Note: In this case, the CsmKey should be configured as a SHE key. The format should be of algorithm type SHE and the *KeyMCryptoKeyGenerationType* should be set to `KEYM_STORED_KEY`.

[SWS_KeyM_00015] ⌈ When KeyM_Update() is called and a KeyMCryptoKeyId is found either by the internal search algorithm or through the provision of the key handler KeyM_KH_Update(), the key generation shall be performed as configured in *KeyMCryptoKeyGenerationType*. If no associated key was found the KeyM_Update() function shall return `E_NOT_OK`.
⌋(SRS_CryptoStack_00119)

[SWS_KeyM_00016] ⌈ If a key ID was identified and *KeyMCryptoKeyGenerationType* is configured as `KEYM_STORED_KEY`, the function `Csm_KeyElementSet()` will be called with the reference to *KeyMCryptoKeyCsmKeyTargetRef* and key element id ‘1’. An internal marker will be set for this key that the contents have been altered and need to be finalized.
⌋(SRS_CryptoStack_00118)

[SWS_KeyM_00017] ⌈ If a key ID was identified and *KeyMCryptoKeyGenerationType* is configured as `KEYM_DERIVE_KEY`, the function `Csm_KeyDerive()` will be called to derive a new key (referenced by *KeyMCryptoKeyCsmKeyTargetRef*) out of the common key (referenced by *KeyMCryptoKeyCsmKeySourceDeriveRef*). An internal

marker will be set for this key that the contents have been altered and need to be finalized.
⌋(SRS_CryptoStack_00118)

[SWS_KeyM_00018] ⌈ If the *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to `FALSE`, the function `Csm_KeySetValid()` shall be called immediately after a successful key derive or store operation.
⌋(SRS_CryptoStack_00118)

There are several options on how to operate key updates:
One obvious option is to call the `KeyM_Update()` function several times, i.e. once per key that shall be updated. The key master will trigger the function call from outside and will provide the key material with every service function. Another possibility is to provide a container with one single call to e.g. `KeyM_Prepare()` which in turn calls `KeyM_KH_Prepare()`. This allows to provide the container in an OEM specific format. The key handler will scan the container and has to call `KeyM_Update()` several times for each key available in the container.

[SWS_KeyM_00019] ⌈ If the configuration item *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to `TRUE`, the crypto key operation has to be concluded with a call to `KeyM_Finalize()`. This function will trigger a call to `Csm_KeySetValid()` for all keys that have an internal marker set to finalize the key update operation. The key update session is closed after this function call and all internal markers are cleared, regardless if the function call was successful or not.
⌋(SRS_CryptoStack_00118, SRS_CryptoStack_00119)

[SWS_KeyM_00020] ⌈ If the configuration item *KeyMCryptoKeyVerifyFunctionEnabled* is set to TRUE, the crypto key submodule shall provide the function `KeyM_Verify()`. This function can be triggered by the key master and is used to run a crypto job referenced by *KeyMCryptoKeyCsmVerifyJobRef*. `KeyM_Verify()` can be called at any time and is not bound to an active crypto key session.
⌋(SRS_CryptoStack_00119)

7.2 Certificate Submodule

The certificate submodule functions of KeyM allow BSW modules and SWCs to perform operations with certificates more efficiently and on a central point within the AUTOSAR software architecture. Examples for such operations are the verification of a complete certificate chain or retrieving elements from a certificate that was provided and verified at runtime.
The required cryptographic operations such as verification of a certificate signature are still performed by associated crypto jobs that are defined in the Crypto Service Manager. Also, the secure storage of certificates can be located in key storage locations of the CSM, e.g. to allow to store the root certificate within the HSM.

7.2.1 General behavior

The certificate submodule allows to define and configure certificates so that they can be stored at production time and further be used for several purposes. The configuration allows to define certificates of a certificate chain in a hierarchical structure with root, intermediate and target certificates used in a PKI system. The stored certificates will be checked at startup according to the configured hierarchy. The configuration allows also to check if specific certificate elements have determined values. There is further support to read specific elements of a certificate and the contained public key can be associated to a CsmKey to use them with configured CSM crypto jobs.

One important part of the specification is therefore the configuration to define the parts of a certificate for flexible and comprehensive verification and for information extraction. The certificates can be associated to *KeyMCryptoKey* container. This allows a permanent storage of certificates in either NVM or CSM.

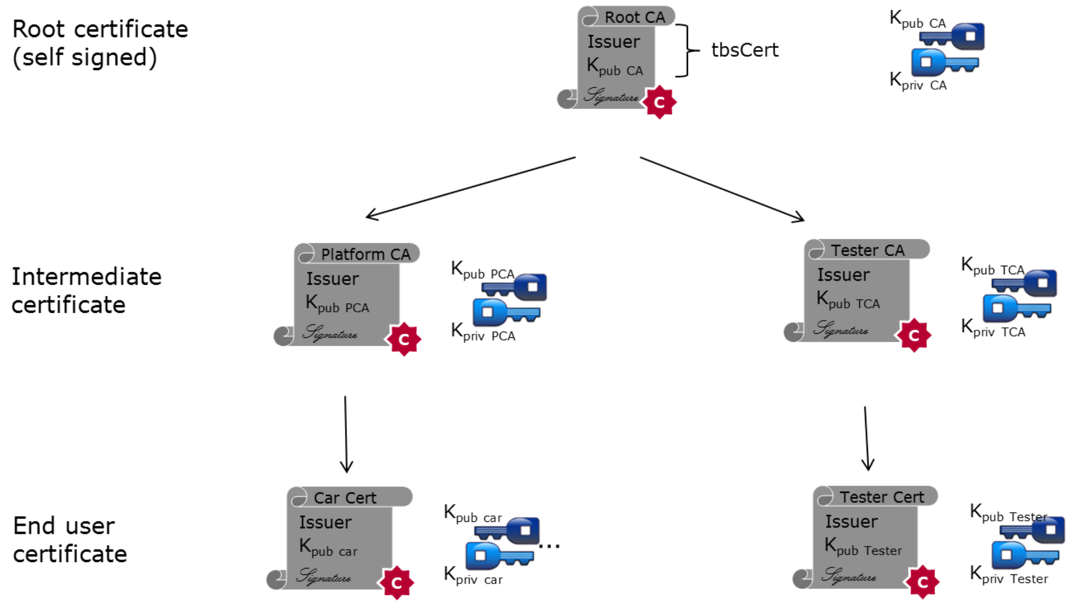


Figure 7-2: Exemplary PKI certificate chain

Root and intermediate certificates, if required, can be provided in the production phase of the ECU or the vehicle. These certificates will be permanently stored in a specified place. If a certificate is now presented to the ECU, this certificate can be stored in a temporary place to request the verification. The certificate submodule will check for existing certificates in the associated chain and will start to parse the contents, verify them against pre-configured conditions and will then check the signatures against all available certificates in its chain.

7.2.2 Initialization

[SWS_KeyM_00022] ⌈ During initialization, the certificate submodule will retrieve the permanently stored certificates, will prepare them for parsing and make them available on demand, e.g. for certificate element extraction or verification against other certificates.
⌋(SRS_CryptoStack_00111, SRS_CryptoStack_00118)

Optionally, instead of parsing the certificate on every startup, the certificate submodule can parse the certificate once and store the parsed information in a dedicated NVM block. The advantage to store parsing results in NVM would lead to faster startup of the system.

Since parsing and verification of certificates can take a significant amount of time it is recommended to perform this operation for stored certificates in the background task after startup.

[SWS_KeyM_00023] ⌈ If the parsing operation was successful, the certificate submodule extracts the public key from the certificate and stores it in the provided key reference of the CSM or in NVM.
⌋(SRS_CryptoStack_00117, SRS_CryptoStack_00118)

7.2.3 Certificate configuration

[SWS_KeyM_00024] ⌈ At least one certificate shall be defined as the Root certificate of a PKI. The *KeyMCertUpperHierarchicalCertRef* of the corresponding *KeyMCertificate* container is referencing to itself.
⌋(SRS_CryptoStack_00111)

Rationale:
A root certificate has the characteristics, that the signature is verified with the public key stored in the same certificate (self-signed certificate). It is the top certificate in the hierarchy.

Figure 7-3 shows a configuration of three *KeyMCertificate* containers in a hierarchal way. It illustrates the configuration of the CSM job and key and the references from the *KeyMCertificate*. This shows, which CSM job and key are referenced by the containers and which job is used for signature verification.

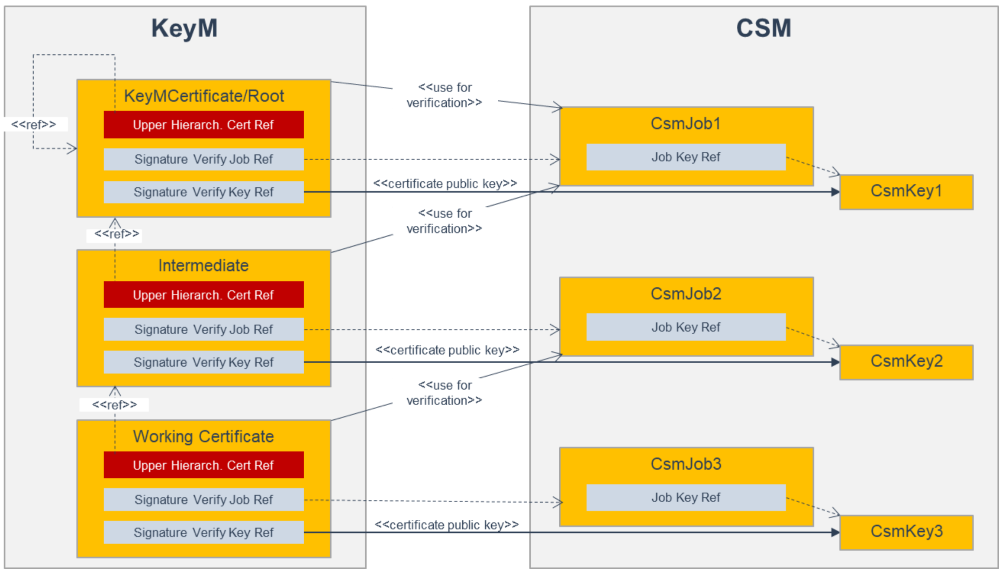


Figure 7-3: Exemplary configuration of a certificate chain in a hierarchy with references to CSM jobs and keys.

[SWS_KeyM_00025] ⌈ A certificate is stored for verification with the call of the function `KeyM_SetCertificate()`. The certificate will be placed in the preconfigured storage class of the *KeyMCertificate/KeyMCertificateStorage*.
⌋(SRS_CryptoStack_00120)

Note:
Such a certificate is typically placed in RAM and is not intended to be used for permanent storage. `KeyM_SetCertificate()` is just used for the verification of a presented certificate. It is not intended to be used for permanent storage like for example the Root certificate. For operation to store a certificate permanently, the function `KeyM_ServiceCertificate()` shall be used.

Certificates can be represented in different formats. The configuration foresees three different formats, the X.509, CVC and CRL. Key elements that are assigned to certificates can be categorized into basic elements of the structure. This is configured with *KeyMCertificateElement/KeyMCertificateElementOfStructure*.

The following tables give correspondences of the enum values of *KeyMCertificateElementOfStructure* to the naming convention of the respective specifications.

X.509	**RFC 5280**	**KeyM Configuration of KeyMCertificateElement/ *KeyMCertificateElementOfStructure***
	Version	CertificateVersionNumber
	Certificate Serial Number	CertificateSerialNumber

	Signature Algorithm Identifier	CertificateSignatureAlgorithmID
	Issuer Name	CertificateIssuerName
	Validity Not Before Time	CertificateValidityPeriodNotBefore
	Validity Not After Time	CertificateValidityPeriodNotAfter
	Subject Name	CertificateSubjectName
	Subject Public Key Algorithm	CertificateSubjectPublicKeyInfo_PublicKeyAlgorithm
	Subject Public Key	CertificateSubjectPublicKeyInfo_SubjectPublicKey
	Extensions	CertificateExtension

Table 7-1: Corresponding items of X.509 elements from RFC5280 with element item configuration of *KeyMCertificateElement/KeyMCertificateElementOfStructure* in [ECUC_KeyM_00038].

	BSI - Technical Guideline TR-03110	**KeyM Configuration of KeyMCertificateElement/ *KeyMCertificateElementOfStructure***
CVC	Certificate Profile Identifier	CertificateVersionNumber
	Certificate Authority Reference	CertificateIssuerName
	Signature Algorithm Identifier	CertificateSignatureAlgorithmID
	Public Key Object Identifier	CertificateSubjectPublicKeyInfo_PublicKeyAlgorithm
	Public Key Domain Parameters	CertificateSubjectPublicKeyInfo_SubjectPublicKey
	Certificate Holder Reference	CertificateSubjectName
	Certificate Holder Authorization Template	CertificateSubjectAuthorization
	Certificate Effective Date	CertificateValidityPeriodNotBefore
	Certificate Expiration Date	CertificateValidityPeriodNotAfter
	Certificate Extensions	CertificateExtension

Table 7-2: Corresponding items of CVC elements as defined in BSI - Technical Guideline TR-03110 with element item configuration of *KeyMCertificateElement/KeyMCertificateElementOfStructure* in [ECUC_KeyM_00038].

The element *RevokedCertificates* in **[ECUC_KeyM_00038]** is used to indicate that this element is the list of revoked certificates in the certificate revocation list (CRL).

7.2.4 Operation mode

[SWS_KeyM_00021] ⌈ If the configuration item *KeyMServiceCertificateFunctionEnabled* and *KeyMCertificateManagerEnabled* is set to TRUE, the certificate submodule shall provide the function `KeyM_ServiceCertificate()`. This function can be triggered by the key master and is used to provide certificate related information to the certificate submodule. Several certificate related operations can be performed like introduction or update of certificates that are permanently stored in the system.
⌋(SRS_CryptoStack_00080)

Every certificate that can be addressed by its symbolic name *KeyMCertificate/KeyMCertificateId* provides a status as defined in `KeyM_CertificateStatusType`. Each status will be entered by a defined state transition that is outlined in Figure 7-4. The current status of a certificate can be requested by `KeyM_CertGetStatus()`.

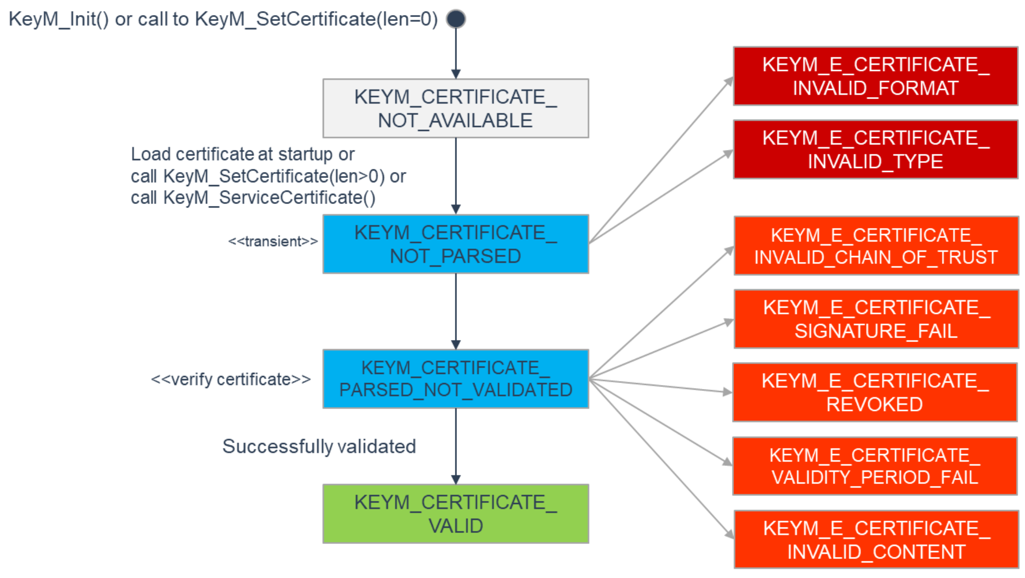

KeyM_Init() or call to KeyM_SetCertificate(len=0)
KEYM_CERTIFICATE_NOT_AVAILABLE
Load certificate at startup or call KeyM_SetCertificate(len>0) or call KeyM_ServiceCertificate()
<<transient>>
KEYM_CERTIFICATE_NOT_PARSED
<<verify certificate>>
KEYM_CERTIFICATE_PARSED_NOT_VALIDATED
Successfully validated
KEYM_CERTIFICATE_VALID
KEYM_E_CERTIFICATE_INVALID_FORMAT
KEYM_E_CERTIFICATE_INVALID_TYPE
KEYM_E_CERTIFICATE_INVALID_CHAIN_OF_TRUST
KEYM_E_CERTIFICATE_SIGNATURE_FAIL
KEYM_E_CERTIFICATE_REVOKED
KEYM_E_CERTIFICATE_VALIDITY_PERIOD_FAIL
KEYM_E_CERTIFICATE_INVALID_CONTENT

Figure 7-4: Certificate status and possible state transitions.

[SWS_KeyM_00167] ⌈ The certificate status "`KEYM_CERTIFICATE_NOT_AVAILABLE`" is entered after initialization of KeyM. This status can also be entered by a call to `KeyM_SetCertificate()` with a length value of 0 in certDataLength within the `KeyM_CertDataType` structure to reset a certificate and its status (see also **[SWS_KeyM_00141]**).
⌋(SRS_CryptoStack_00115)

[SWS_KeyM_00026] ⌈ The parsing process of a certificate shall be started as soon as the certificate has been stored with either `KeyM_SetCertificate()` or `KeyM_ServiceCertificate()`. When parsing is in progress, the certificate status shall change to the transient status `KEYM_CERTIFICATE_NOT_PARSED` until the parsing process is completed.

The parsing process can also be already triggered on initialization of KeyM, as outlined in **[SWS_KeyM_00022]**. In the same way, the certificate status shall change to `KEYM_CERTIFICATE_NOT_PARSED`.
⌋(SRS_CryptoStack_00117)

[SWS_KeyM_00027] ⌈ ⌈ If the certificate parse operation detects violations to basic encoding rules on the *KeyMCertFormatType* such as ASN.1 or TLV (Tag-Length-Values) or if basic elements for that *KeyMCertFormatType* are missing, the certificate status `KEYM_CERTIFICATE_INVALID_FORMAT` shall be set and reported to the application if required by configured callback. No further operation like parsing and validating shall be performed on this certificate until the status has been reset.
⌋(SRS_CryptoStack_00111)

[**SWS_KeyM_00168**] ⌈ If the certificate is in a well-formatted ASN.1 structure but basic elements as outlined in `KeyMCertificateElementOfStructure` for the specified format type (*KeyMCertFormatType*) are missing, the certificate status `KEYM_E_CERTIFICATE_INVALID_TYPE` shall be set and reported to the application if required by the configured callback. No further operation like validating shall be performed on this certificate until the status has been reset.

⌋(SRS_CryptoStack_00111)

[**SWS_KeyM_00169**] ⌈ If the parsing operation has been completed without failure, the certificate status shall be set to `KEYM_CERTIFICATE_PARSED_NOT_VALIDATED` and reported to the application if required by the configured callback.

⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00028] ⌈ A verification of a certificate shall only be started if the certificate is in the status `KEYM_CERTIFICATE_PARSED_NOT_VALIDATED`. The verification shall be triggered at the latest by a call to one of the functions `KeyM_VerifyCertificate(),KeyM_VerifyCertificates()` or `KeyM_VerifyCertificateChain().`
⌋(SRS_CryptoStack_00111)

Note:
It is up to the implementation of a KeyM to start the verification process earlier than by one of these function calls, e.g. in the background after initialization or if a certificate chain needs to be verified.

[SWS_KeyM_00029] ⌈ The verification of a certificate shall be done in sequential steps as described in the following requirements. If one step fails, the certificate status shall be set to the corresponding value, it shall be reported through a callback function (if configured) and the verification shall be stopped. The certificate status shall remain in this status until it is reset as outlined in **[SWS_KeyM_00167]**.
⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00030] ⌈ The verification of a certificate starts with a check if all certificates that are linked with *KeyMCertUpperHierarchicalCertRef* are in the status `KEYM_CERTIFICATE_VALID`.
If either of these referenced certificates are in the status `KEYM_CERTIFICATE_NOT_PARSED` or `KEYM_CERTIFICATE_PARSED_NOT_VALIDATED`, the parse and/respectively verification process shall be started for these linked certificates in the order from top (the last one of the linked list that either has no further link or links to itself) down to the bottom (the next certificate that is directly linked with *KeyMCertUpperHierarchicalCertRef* of the currently processed certificate). In this case, the status check of the linked certificates shall be re-done after all initiated certificate verifications have reached a final status.

⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00031] ⌈ If all certificates that are linked with *KeyMCertUpperHierarchicalCertRef* are in the status `KEYM_CERTIFICATE_VALID`, or *KeyMCertUpperHierarchicalCertRef* links to itself (self-signed certificates), the subject field of the currently validated certificate shall be verified with the issuer field of the next upper certificate (the one referenced by *KeyMCertUpperHierarchicalCertRef*).
If one of the checks above have failed, the status `KEYM_E_CERTIFICATE_INVALID_CHAIN_OF_TRUST` shall be set and reported through callback function (if configured) and the validation process shall stop.
Otherwise, the check outlined in **[SWS_KEYM_00032]** shall be performed at next.

⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00032] ⌈ The signature of the certificate shall be verified by using the CSM verify job referenced with *KeyMCertUpperHierarchicalCertRef/ KeyMCertCsmSignatureVerifyJobRef* (see Figure 7-4). It should be noted, that for self-signed certificates, the public key of this certificate needs to be set and validated in *KeyMCertCsmSignatureVerifyKeyRef* beforehand.
If the signature verification fails, the certificate status `KEYM_E_CERTIFICATE_SIGNATURE_FAIL` shall be set and reported through callback function (if configured) and the validation process shall stop.
Otherwise, the check outlined in **[SWS_KEYM_00033]** shall be performed at next.

⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00033] ⌈ If the KeyM module maintains revocation lists, it shall check if the certificate under validation is part of the revoked one. If so, the certificate status `KEYM_E_CERTIFICATE_REVOKED` shall be set and reported through callback function (if configured) and the validation process shall stop.
Otherwise, the check outlined in **[SWS_KEYM_00034]** shall be performed at next.

⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00034] ⌈ If the certificate format type contains a time period, the KeyM module shall query the current time from configured time source (*KeyMCertTimebaseRef*) and compare the current time if it is within the validity

period of the certificate. If not, the certificate status `KEYM_E_CERTIFICATE_VALIDITY_PERIOD_FAIL` shall be set and reported through callback function (if configured) and the validation process shall stop. Otherwise, the check outlined in **[SWS_KEYM_00035]** shall be performed at next.
⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00035] ⌈ The contents of certificate elements shall be checked through by a check of all *KeyMCertCertificateElementRuleRef*. If one of the rules are violated, the certificate status `KEYM_E_CERTIFICATE_INVALID_CONTENT` shall be set and reported through callback function (if configured) and the validation process shall stop.
Otherwise, the requirement outlined in **[SWS_KeyM_00170]** shall be performed.
⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00170] ⌈ If all verification steps have been performed and no error was detected during the verification, the public key of the certificate shall be set and validated (`Csm_KeySetValid()`) to the CSM key referenced by *KeyMCertCsmSignatureVerifyKeyRef* (if not already done for self-signed certificates, see **[SWS_KeyM_00032]**). The certificate status `KEYM_CERTIFICATE_VALID` shall be set and reported to the application if required by configured callback.
⌋(SRS_CryptoStack_00118)

7.3 Security Events

[**SWS_KeyM_00171**] ⌈ If security event reporting has been enabled for the KeyM module (*KeyMEnableSecurityEventReporting* = true) the respective security events shall be reported to the IdsM via the interfaces defined in AUTOSAR_SWS_BSWGeneral.
⌋(RS_Ids_00810, SRS_CryptoStack_00121)

The following table lists the security events which are standardized for the KeyM together with their trigger conditions:

[SWS_KeyM_00172]⌈

Name	***Description***	***ID***
KEYM_SEV_INST_ROOT_CERT_OP	Attempt to install a root certificate.	1
KEYM_SEV_UPD_ROOT_CERT_OP	Attempt to update an existing root certificate.	2
KEYM_SEV_INST_INTERMEDIATE_CERT_OP	Attempt to install an intermediate certificate.	3
KEYM_SEV_UPD_INTERMEDIATE_CERT_OP	Attempt to update an intermediate certificate.	4

KEYM_SEV_CERT_VERIF_FAILED	A request to verify a certificate against a certificate chain was not successful.	5

⌋(RS_Ids_00810)

[**SWS_KeyM_00173**] ⌈ The following table describes the context data which shall be reported for the respective security event:

Security Event	**Context Data**
KEYM_SEV_INST_ROOT_CERT_OP	Context Data (41 Byte) • Result (1 Byte) ○ Operation failed: 0x0 ○ Operation succeeded: 0x1 • HashedID8 of certificate (8 Byte) • certificateIssuerName (32 Byte)
KEYM_SEV_UPD_ROOT_CERT_OP	Context Data (41 Byte) • Result (1 Byte) ○ Operation failed: 0x0 ○ Operation succeeded: 0x1 • HashedID8 of certificate (8 Byte) • certificateIssuerName (32 Byte)
KEYM_SEV_INST_INTERMEDIATE_CERT_OP	Context Data (41 Byte) • Result (1 Byte) ○ Operation failed: 0x0 ○ Operation succeeded: 0x1 • HashedID8 of certificate (8 Byte) • certificateIssuerName (32 Byte)
KEYM_SEV_UPD_INTERMEDIATE_CERT_OP	Context Data (41 Byte) • Result (1 Byte) ○ Operation failed: 0x0 ○ Operation succeeded: 0x1 • HashedID8 of certificate (8 Byte) • certificateIssuerName (32 Byte)
KEYM_SEV_CERT_VERIF_FAILED	Context Data (41 Byte) • Result (1 Byte) ○ E_NOT_OK: 0x0 ○ KEYM_E_BUSY: 0x1 ○ KEYM_E_PARAMETER_MISMATCH: 0x05 ○ KEYM_E_KEY_CERT_EMPTY: 0x0A ○ KEYM_E_CERT_INVALID_CHAIN_OF_TRUST: 0x0B • HashedID8 of certificate (8 Byte) • certificateIssuerName (32 Byte)

⌋(RS_Ids_00810, SRS_CryptoStack_00121)

7.4 Error classification

Section 7.2 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.4.1 Development Errors

[SWS_KeyM_00036]⌈

Type of error	*Related error code*	*Error value*
API service called with invalid parameter (Null Pointer)	KEYM_E_PARAM_POINTER	0x01
Buffer is too small for operation	KEYM_E_SMALL_BUFFER	0x02
API called before module has been initialized	KEYM_E_UNINIT	0x03
KeyM module initialization failed	KEYM_E_INIT_FAILED	0x04
KeyM configuration failure	KEYM_E_CONFIG_FAILURE	0x05

⌋()

7.4.2 Runtime Errors

There are no runtime errors.

7.4.3 Transient Faults

There are no transient faults.

7.4.4 Production Errors

There are no production errors.

7.4.5 Extended Production Errors

There are no extended production errors.

7.5 Error detection

[SWS_KeyM_00144] ⌈ If development errors are active the Key Manager shall check on every function call if the module has been initialized with KeyM_Init() and not yet been de-initialized with KeyM_Deinit(). Otherwise, the Development error KEYM_E_UNINIT shall be set.
⌋(SRS_CryptoStack_00087)

[SWS_KeyM_00145] ⌈ If development errors are active the Key Manager shall check on every function where result buffers are provided if the provided buffer is large enough to store the requested result. If not, the development error KEYM_E_SMALL_BUFFER shall be set.
⌋(SRS_CryptoStack_00087)

[SWS_KeyM_00146] ⌈ If development errors are active the Key Manager shall check on every function where pointers are provided if the pointer is not a NULL_PTR. If a NULL_PTR is provided but not expected, the development error KEYM_E_PARAM_POINTER shall be set.
⌋(SRS_CryptoStack_00087)

8 API specification

8.1 Imported types

In this chapter all types included from the following files are listed:

[SWS_KeyM_00037]⌈

Module	***Header File***	***Imported Type***
Csm	Rte_Csm_Type.h	Crypto_OperationModeType
	Rte_Csm_Type.h	Crypto_VerifyResultType
IdsM	IdsM_Types.h	IdsM_SecurityEventIdType
StbM	Rte_StbM_Type.h	StbM_SynchronizedTimeBaseType
	Rte_StbM_Type.h	StbM_TimeBaseStatusType
	Rte_StbM_Type.h	StbM_TimeStampType
	Rte_StbM_Type.h	StbM_UserDataType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋()

8.2 Type definitions

8.2.1 KeyM_ConfigType

[SWS_KeyM_00157]⌈

Name	KeyM_ConfigType

Kind	Structure	
Elements	Implementation specific	
	Type	--
	Comment	The content of this data structure is implementation specific
Description	This structure is the base type to initialize the Key Manager module. A pointer to an instance of this structure will be used in the initialization of the Key Manager module.	
Available via	KeyM.h	

⌋(SRS_BSW_00404)

8.2.2 KeyM_KH_UpdateOperationType

[SWS_KeyM_00055]⌈

Name	KeyM_KH_UpdateOperationType		
Kind	Enumeration		
Range	KEYM_KH_ UPDATE_KEY_ UPDATE_REPEAT	0x01	Key handler has successfully performed the operation and provides new key data that shall be further operated by the update function of the key manager. A next call to key handler is requested.
	KEYM_KH_ UPDATE_FINISH	0x02	Key handler has successfully performed all update operation. The update operation is finished and the result data can be provided back for a final result of the KeyM_ Update operation.
Description	Specifies the type of key handler update operation that was performed in the callback.		
Available via	KeyM.h		

⌋()

8.2.3 KeyM_CertElementIteratorType

[SWS_KeyM_00042]⌈

Name	KeyM_CertElementIteratorType	
Kind	Structure	
Elements	Implementation specific	
	Type	--
	Comment	The content of this data structure is implementation specific
Description	This structure is used to iterate through a number of elements of a certificate.	
Available via	KeyM.h	

⌋()

8.2.4 KeyM_CryptoKeyIdType

[SWS_KeyM_00302]⌈

Name	KeyM_CryptoKeyIdType
Kind	Type
Derived from	uint16
Description	Crypto key handle.
Available via	KeyM.h

⌋()

8.2.5 KeyM_CertDataPointerType

[SWS_KeyM_91011]⌈

Name	KeyM_CertDataPointerType

Kind	Pointer
Type	uint8*
Description	Byte-pointer to the data of the certificate
Available via	KeyM.h

⌋()

8.2.6 Extension to Std_ReturnType

The Key Management module uses the following extension to the Std_ReturnType:
[SWS_KeyM_00040]⌈

Range	KEYM_E_BUSY	0x02	Key management is busy with other operations.
	KEYM_E_PENDING	0x03	Operation request accepted, response is pending. It runs now in asynchronous mode, response will be given through callback.
	KEYM_E_KEY_CERT_SIZE_MISMATCH	0x04	Parameter size does not match the expected value.
	KEYM_E_PARAMETER_MISMATCH	0x05	Parameter to function does not provide the expected value.
	KEYM_E_KEY_CERT_INVALID	0x06	Key or certificate is invalid and cannot be used for the operation.
	KEYM_E_KEY_CERT_WRITE_FAIL	0x07	Certificate or key write operation failed.
	KEYM_E_KEY_CERT_UPDATE_FAIL	0x08	Key or certificate update operation failed.
	KEYM_E_KEY_CERT_READ_FAIL	0x09	Certificate or key could not be provided due to a read or permission failure.
	KEYM_E_KEY_CERT_EMPTY	0x0A	The requested key or certificate is not available, slot is empty.

	KEYM_E_CERT_ INVALID_CHAIN_OF_ TRUST	0x0B	Certificate verification failed - Invalid Chain of Trust
Description	Key management specific return values for use in Std_ReturnType.		
Available via	KeyM.h		

⌋()

8.3 Function definitions

This is a list of functions provided to upper layer modules.

8.3.1 General

8.3.1.1 KeyM_Init

[SWS_KeyM_00043]⌈

Service Name	KeyM_Init	
Syntax	`void KeyM_Init ( const KeyM_ConfigType* ConfigPtr )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to the configuration set in VARIANT-POST-BUILD.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This function initializes the key management module.	
Available via	KeyM.h	

⌋(SRS_BSW_00101, SRS_BSW_00358, SRS_BSW_00414)

[SWS_KeyM_00158] ⌈ The Configuration pointer configPtr shall always have a `NULL_PTR` value.
⌋(SRS_BSW_00414)

Note: A Configuration of the Key Manager at initialization is currently not used and shall therefore pass a `NULL_PTR` to the module.

[SWS_KeyM_00044] ⌈ If the initialization of the key management module fails and development errors are activated, the error `KEYM_E_INIT_FAILED` shall be reported to the DET.
⌋ (SRS_CryptoStack_00087)

[SWS_KeyM_00045] ⌈ If the certificate submodule is active and permanently stored certificates are available in unparsed and unverified state, the KeyM certificate submodule part shall start a background task to pre-parse and pre-verify certificates.
⌋(SRS_CryptoStack_00031, SRS_CryptoStack_00111)

Rationale: The operation can be done in a background task if CPU time is available, Pre-validating certificates will help to speed-up the authentication when a certificate is presented and shall be verified at runtime against a pre-installed certificate chain.

[SWS_KeyM_00046] ⌈ If the crypto key submodule is active, all keys stored in NVM shall be read from and stored to CSM (RAM-) key slots during initialization.
⌋(SRS_CryptoStack_00114, SRS_CryptoStack_00118)

8.3.1.2 KeyM_Deinit

[SWS_KeyM_00047]⌈

Service Name	KeyM_Deinit

Syntax	`void KeyM_Deinit (`
	`  void`
	`)`

Service ID [hex]	0x02
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None

Parameters (out)	None
Return value	None
Description	This function resets the key management module to the uninitialized state.
Available via	KeyM.h

⌋()

[SWS_KeyM_00048] ⌈
For security reason the crypto key submodule shall actively destroy all data in RAM that was used for cryptographical key material. Especially symmetric keys and intermediate results shall be set to an initial value.
⌋(SRS_CryptoStack_00120)

8.3.1.3 KeyM_GetVersionInfo

[SWS_KeyM_00049]⌈

Service Name	KeyM_GetVersionInfo	

Syntax	`void KeyM_GetVersionInfo (`
	`  Std_VersionInfoType* VersionInfo`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	VersionInfo	Pointer to the version information of this module.
Return value	None	
Description	Provides the version information of this module.	
Available via	KeyM.h	

⌋(SRS_BSW_00407)

8.3.2 Crypto key operation

8.3.2.1 KeyM_Start

[SWS_KeyM_00050]⌈

Service Name	KeyM_Start	
Syntax	`Std_ReturnType KeyM_Start ( KeyM_StartType StartType, const uint8* RequestData, uint16 RequestDataLength, uint8* ResponseData, uint16* ResponseDataLength )`	
Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	StartType	Defines in which mode the key operation shall be executed.
	RequestData	Information that comes along with the request, e.g. signature
	RequestData Length	Length of data in the RequestData array
Parameters (inout)	ResponseData Length	In: Max number of bytes available in ResponseData Out: Actual number
Parameters (out)	ResponseData	Data returned by the function.
Return value	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 do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match

Description	This function is optional and only used if the configuration item KeyMCryptoKey StartFinalizeFunctionEnabled is set to true. It intents to allow key update operation.
Available via	KeyM.h

⌋()

[SWS_KeyM_00085] ⌈ If *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to `TRUE`, this function shall be called to initiate a key update session. The function indicates with `E_OK` that key operations are now possible.
⌋(SRS_CryptoStack_00028, SRS_CryptoStack_00096)

[SWS_KeyM_00086] ⌈ If a key update session is already active and the function is called with the same parameter, this function shall return with `E_OK` and continue to accept key update operations.
⌋(SRS_CryptoStack_00028, SRS_CryptoStack_00096)

[SWS_KeyM_00087] ⌈ By default, the `KeyM_Start()` function does not check RequestData length or values. It will accept every function call with valid startTypes to initiate key update sessions.
⌋(SRS_CryptoStack_00107)

[SWS_KeyM_00088] ⌈ OEM or security specific checks for the start operation shall be performed in the corresponding key handler operation.
⌋(SRS_CryptoStack_00115)

8.3.2.2 KeyM_Prepare

[SWS_KeyM_00051]⌈

Service Name	KeyM_Prepare

Syntax	`Std_ReturnType KeyM_Prepare (`
	`  const uint8* RequestData,`
	`  uint16 RequestDataLength,`
	`  uint8* ResponseData,`
	`  uint16* ResponseDataLength`
	`)`

Service ID [hex]	0x05
Sync/Async	Synchronous
Reentrancy	Non Reentrant

Parameters (in)	RequestData	Information that comes along with the request
	RequestData Length	Length of data in the RequestData array
Parameters (inout)	ResponseData Length	In: Max number of bytes available in ResponseData Out: Actual number of bytes
Parameters (out)	ResponseData	Data returned by the function.
Return value	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 do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match
Description	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.	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00089] ⌈ The function `KeyM_Prepare()` is provided when *KeyMCryptoKeyPrepareFunctionEnabled* is set to `TRUE`. There is no dedicated implementation, but a key handler can be used to provide specific information to the key server that is required to generate key material. Such information or further operation can be performed through the key handler callback `KeyM_KH_Prepare()` when enabled, e.g. providing SHE information or generating secret key generation operations.
⌋(SRS_CryptoStack_00107, SRS_CryptoStack_00109, SRS_CryptoStack_00115, SRS_CryptoStack_00027)

[SWS_KeyM_00090] ⌈ By default, the function returns `E_NOT_OK`. If a key handler is configured to be called, this function will call the key handler with the exact parameter and will pass the return value of this key handler back to the caller.
⌋(SRS_CryptoStack_00096)

8.3.2.3 KeyM_Update

[SWS_KeyM_00052]⌈

Service Name	KeyM_Update	
Syntax	`Std_ReturnType KeyM_Update ( const uint8* KeyNamePtr, uint16 KeyNameLength, const uint8* RequestDataPtr, uint16 RequestDataLength, uint8* ResultDataPtr, uint16 ResultDataMaxLength )`	
Service ID [hex]	0x06	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	KeyNamePtr	Pointer to an array that defines the name of the key to be updated
	KeyName Length	Specifies the number of bytes in keyName. The value 0 indicates that no keyName is provided within this function.
	RequestData Ptr	Information that comes along with the request
	RequestData Length	Length of data in the RequestData array
	ResultDataMax Length	Max number of bytes available in ResultDataPtr.
Parameters (inout)	None	
Parameters (out)	ResultDataPtr	Pointer to a data buffer used by the function to store results.
Return value	Std_Return-Type	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 do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match

Description	This function is used to initiate the key generation or update process.
Available via	KeyM.h

⌋()

By the call of this function a key update operation is requested.

[SWS_KeyM_00091] ⌈ If a KeyName is provided the Key Manager shall search for an element in the container that matches */KeyMCryptoKey/KeyMCryptoKeyName*. If found, the CryptoKeyId shall be used as KeyID and this container shall be used for reference of any further key update operation (derive or store the key value).
⌋(SRS_CryptoStack_00113, SRS_CryptoStack_00105, SRS_CryptoStack_00027, SRS_CryptoStack_00010)

[SWS_KeyM_00154] ⌈ If either KeyNamePtr is not valid or KeyNameLength is 0 and *KeyMCryptoKeyCryptoProps* is defined then the Key Manager shall interpret the RequestData as M1M2M3 values of a SHE key. The Key Manager shall extract bits 121..124 located in RequestDataPtr (if RequestDataLength indicates enough data) and shall check for a corresponding value in *KeyMCryptoKeyCryptoProps*. If a matching value is found then *CryptoKeyId* of this container shall be used as KeyID and this container shall be used for reference of any further key update operation (derive or store the key value).
⌋(SRS_CryptoStack_00114)

[SWS_KeyM_00155] ⌈ If a KeyID could not be identified and *KeyMCryptoKeyHandlerUpdateEnabled* is set to `FALSE` then `KeyM_Update()` shall not perform a key update operation and shall return `KEYM_E_PARAMETER_MISMATCH`.
⌋(SRS_CryptoStack_00086, SRS_CryptoStack_00096)

[SWS_KeyM_00092] ⌈ If *KeyMCryptoKeyHandlerUpdateEnabled* is set to `TRUE` to perform a key handler operation then `KeyM_Update()` shall call `KeyM_KH_Update()`. The parameter RequestDataPtr, RequestDataLength, KeyName and KeyNameLength shall be passed on to the key handler. If a *KeyMCryptoKey* container was identified in one of the previous steps then the *KeyMCryptoKeyID* shall be provided with the KeymId parameter. Otherwise, the value 0xFFFFFFFFul shall be used.
⌋(SRS_CryptoStack_00107)

[SWS_KeyM_00098] ⌈ If no key handler is configured for the key update operation (*KeyMCryptoKeyHandlerUpdateEnabled* is set to `FALSE`) and a CryptoKey container was identified, a key update operation shall be performed according to the configuration (derive or store key in CSM). stored according to the configuration. Thus, if *KeyMCryptoKeyStorage* is set to `KEYM_STORAGE_IN_NVM` is set, the ResultData and length for this key ID shall be stored in the configured NVM block.

Otherwise, if KEYM_STORAGE_IN_CSM is set, the CSM is responsible to store the key data after it has been set.
⌋(SRS_CryptoStack_00117, SRS_CryptoStack_00118)

[SWS_KeyM_00099] ⌈ If a key was identified by its ID and either RequestDataPtr and RequestDataLength indicates data or KeyM_KH_Update() has returned E_OK and ResultDataPtr and ResultDataLengthPtr indicates data and the configuration */KeyMCryptoKey/KeyMCryptoKeyGenerationType* is set to KEYM_STORED_KEY, then this function shall call Csm_KeyElementSet() to provide the data to CSM. The key element ID is always 1 and the *KeyMCryptoKeyCsmKeyTargetRef* is used to identify the target key.
⌋(SRS_CryptoStack_00096, SRS_CryptoStack_00107)

[SWS_KeyM_00100] ⌈ If a *CryptoKey* container was found and either RequestDataPtr and RequestDataLength provides data or KeyM_KH_Update() has returned E_OK and ResultDataPtr and ResultDataLengthPtr provides data and the configuration */KeyMCryptoKey/KeyMCryptoKeyGenerationType* is set to KEYM_DERIVE_KEY, then the data shall be set to the key element CRYPTO_KE_KEYDERIVATION_PASSWORD. If the configuration value *KeyMCryptoKeyGenerationInfo* is set, then this value shall be used as the salt for the target key and shall set the value to the key element ID CRYPTO_KE_KEYDERIVATION_SALT. The *KeyMCryptoKeyCsmKeyTargetRef* is used to identify the target key and *KeyMCryptoKeyCsmKeySourceDeriveRef* as the source key for the derivation and the function Csm_KeyDerive() shall be called accordingly.
⌋(SRS_CryptoStack_00107, SRS_CryptoStack_00027)

[SWS_KeyM_00101] ⌈ If a key update operation was successful and *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to FALSE, then the function Csm_KeySetValid() shall be called immediately after the key element has been successfully set in CSM.
⌋(SRS_CryptoStack_00107)

[SWS_KeyM_00102] ⌈ If a key update operation was successfully performed through CSM operation and *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to TRUE, then a flag shall be set for this key to indicate, that Csm_KeySetValid() for the key shall be called during finalization of the key update operation.
⌋(SRS_CryptoStack_00107)

[SWS_KeyM_00094] ⌈ KeyM_Update() runs in asynchronous mode. Note that the key handler KeyM_KH_Update() is called in synchronous mode. It shall be called therefore from within the background task.
⌋(SRS_CryptoStack_00101)

[SWS_KeyM_00095] ⌈ If a single key update operation was finished with success or a key update operation has failed because a function call to CSM or key handler has not returned `E_OK` or `KeyM_KH_Update()` has provided the operation type `KEYM_KH_UPDATE_FINISH`, the callback function `KeyM_CryptoKeyUpdateCallbackNotification()` has to be called.
⌋(SRS_CryptoStack_00106, SRS_CryptoStack_00107)

[SWS_KeyM_00156] ⌈ The function that calls `KeyM_Update()` shall provide a pointer to a buffer with ResultDataPtr. If `KeyM_Update()` accepts the operation by returning `E_OK` the function shall not touch this buffer until the callback notification `KeyM_CryptoKeyUpdateCallbackNotification()` has been called. Any results from the `KeyM_Update()` operation will be copied into this buffer. The same buffer pointer provided with the call to `KeyM_Update()` (ResultDataPtr) will be provided as ResultDataPtr with the callback notification. The callback also indicates the length of the result data and the overall result of the update operation.
⌋(SRS_CryptoStack_00107, SRS_CryptoStack_00106)

Info:
The result data is either the result from the key handler or, if no key handler is used, contains the M4M5 for a SHE key.

8.3.2.4 KeyM_Finalize

[SWS_KeyM_00053]⌈

Service Name	KeyM_Finalize	

Syntax	`Std_ReturnType KeyM_Finalize (`
	`  const uint8* RequestDataPtr,`
	`  uint16 RequestDataLength,`
	`  uint8* ResponseDataPtr,`
	`  uint16 ResponseMaxDataLength`
	`)`	

Service ID [hex]	0x07	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	RequestDataPtr	Information that comes along with the request
	RequestData Length	Length of data in the RequestData array

Parameters (inout)	ResponseMax DataLength	In: Max number of bytes available in ResponseData Out: Actual number of bytes in ResponseData or left untouched if service runs in asynchronous mode and function returns KEYM_E_OK.
Parameters (out)	ResponseData Ptr	Data returned by the function.
Return value	Std_Return-Type	E_OK: Operation has been accepted and will be processed internally. Results will be provided 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. KEYM_E_PARAMETER_MISMATCH: Parameter do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match
Description	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 KeyMCryptoKey StartFinalizeFunctionEnabled is set to TRUE. In addition, updated key material will be persisted and set into valid state (calling Csm_KeySetValid).	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00103] ⌈ If *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to TRUE, this function will conclude the key update operation. All keys that have flagged to be updated during the session shall be finalized by calling `Csm_KeySetValid()`.
The validation shall be done for all keys that have been updated, even if `Csm_KeySetValid()` returns a failure for one of the keys. This is to finalize as much keys as possible even if one key fails. If at least one key fails, then the overall result is a fail information in the callback result.
⌋(SRS_CryptoStack_00107)

[SWS_KeyM_00104] ⌈ If *KeyMCryptoKeyStartFinalizeFunctionEnabled* and *KeyMCryptoKeyHandlerStartFinalizeEnabled* is set to `TRUE` this function will call `KeyM_KH_Finalize()` with the exact same parameter as provided with `KeyM_Finalize()`. The finalize key handler has to be called BEFORE the validation of the key (the call to `Csm_KeySetValid()`). If the key handler returns `E_OK`, then this function will continue its operation as specified. If the key handler finalization function returns `E_NOT_OK`, then no validation shall be done.
⌋(SRS_CryptoStack_00096, SRS_CryptoStack_00107)

[SWS_KeyM_00105] ⌈ The callback function `KeyM_CryptoKeyFinalizeCallbackNotification()` will be called if the operation has finished. The parameter 'ResultDataPtr' of this callback shall provide the buffer pointer 'ResponseDataPtr' provided with the call to `KeyM_Finalize()`. The result information provides the residual result of the validation of all keys.
⌋(SRS_CryptoStack_00106)

Info:
Since key validation can take considerable amount of time this function is used in asynchronous mode only. Since the key handler is called in synchronous mode it is recommended to call it not from within `KeyM_Finalize()` but delegate the call to the background task.
The caller of `KeyM_Finalize()` shall provide a buffer that is large enough to store the response. This buffer shall not be touched by the caller if `KeyM_Finalize()` returns `E_OK` until the callback notification has indicated the end of the finalize operation.

[SWS_KeyM_00106] ⌈ At the end of a key finalize operation, all flags for key validation have to be cleared and the session state shall be set to the init mode. Thus, no further key update operations are allowed anymore.
⌋(SRS_CryptoStack_00120)

8.3.2.5 KeyM_Verify

[SWS_KeyM_00054]⌈

Service Name	KeyM_Verify	

Syntax	`Std_ReturnType KeyM_Verify (`
	`  const uint8* KeyNamePtr,`
	`  uint16 KeyNameLength,`
	`  const uint8* RequestData,`
	`  uint16 RequestDataLength,`
	`  uint8* ResponseData,`
	`  uint16* ResponseDataLength`
	`)`	

Service ID [hex]	0x08	
Sync/Async	Synchronous Synchronous/Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	KeyNamePtr	Points to an array that defines the name of the key to be updated
	KeyName Length	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
	RequestData Length	Length of data in the RequestData array
Parameters (inout)	Response DataLength	In: Max number of bytes available in ResponseData Out: Actual number of bytes in ResponseData or left untouched if service runs in asynchronous mode and function returns KEYM_E_PENDING
Parameters (out)	Response Data	Data returned by the function.
Return value	Std_Return-Type	KEYM_E_PENDING: Operation runs in asynchronous mode, has been accepted and will be processed internally. Results will be provided through callback E_OK: Operation was successfully performed. Result information are available. 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 (for asynchronous mode). KEYM_E_PARAMETER_MISMATCH: Parameter do 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.
Description	The key server requests to verify the provided keys. The key manager performs operation on the assigned job and returns the result to the key server who verifies if the results was provided with this key as expected. This function is only available if KeyMCryptoKeyVerifyFunctionEnabled is set to TRUE.	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00107] ⌈ If *KeyMCryptoKeyVerifyFunctionEnabled* is set to `TRUE` this function is available to perform a verification of a key. This function can always be called and is not bound to a key update session.
⌋(SRS_CryptoStack_00119)

[SWS_KeyM_00108] ⌈ If *KeyMCryptoKeyVerifyAsyncMode* is set to `FALSE`, the function will use *KeyMCryptoKey/KeyMCryptoKeyCsmKeyVerifyJobRef* to perform a crypto operation. If specified then the configuration KeyMCryptoCsmVerifyJobType shall be specified as well to identify which job shall be called.
⌋(SRS_CryptoStack_00119, SRS_CryptoStack_00022)

Info:
Since only one input and output buffer is specified, only MAC generate and data decrypt/encrypt operations can be done autonomously in this function. Other operations such as AEAD encrypt/decrypt or MAC verify requires interpretation of structured RequestData which needs to be interpreted in the key handler verification function.

[SWS_KeyM_00109] ⌈ If *KeyMCryptoKeyVerifyAsyncMode* is set to `TRUE`, the function will run in asynchronous mode. The direct function call will return `KEYM_E_PENDING` if the job was accepted or any other return value if the job could not be accepted.
In asynchronous mode, the KeyM_CryptoKeyVerifyCallbackNotification will provide the result of the crypto job operation.
⌋(SRS_CryptoStack_00106, SRS_CryptoStack_00101)

Info:
This is especially useful if at least one CSM verify job is configured for asynchronous operation. Ideally, the verification is initiated in the background task.

8.3.3 Certificate handling

8.3.3.1 KeyM_ServiceCertificate

[SWS_KeyM_00056]⌈

Service Name	KeyM_ServiceCertificate	

Syntax	`Std_ReturnType KeyM_ServiceCertificate (`
	`  KeyM_ServiceCertificateType Service,`
	`  const uint8* CertNamePtr,`
	`  uint16 CertNameLength,`
	`  const uint8* RequestData,`
	`  uint16 RequestDataLength,`
	`  uint8* ResponseData,`
	`  uint16 ResponseDataLength`
	`)`	

Service ID [hex]	0x09	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
	Service	Provides the type of service the key manager has to perform.

Parameters (in)	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
	RequestData Length	Length of data in the RequestData array
	ResponseData Length	Max number of bytes available in ResponseDataPtr.
Parameters (inout)	None	
Parameters (out)	ResponseData	Data returned by the function.
Return value	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 do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match KEYM_E_BUSY Certificate service cannot be executed, operation is busy.
Description	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.	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00110] ⌈ If *KeyMServiceCertificateFunctionEnabled* is set to `TRUE`, this service function is provided to update certificates or certificate information. The type of operation is specified by the Service parameter.
⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00111] ⌈ A service certificate key handler can be configured to defer the service operation. If *KeyMCryptoKeyHandlerServiceCertificateEnabled* is set to TRUE, this function will directly call the service certificate key handler by passing the exact parameter to the handler. It will also return the value returned by the handler and no further operation will be performed.

⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00112] ⌈ If *KeyMCryptoKeyHandlerServiceCertificateEnabled* is set to `FALSE`, the service certificate function will check for the requested service and will perform the requested operation by first searching for a configured certificate by its name.
⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00113] ⌈ Depending on the Service parameter the following services shall be offered:

KEYM_SERVICE_CERT_REQUEST_CSR	Key server requests a certificate signing request. Service certificate shall generate a certificate according to the format, will generate a key pair, either as RSA or ECC, and will store the values in the configured container. The generated certificate will be provided to the key server.
KEYM_SERVICE_CERT_UPDATE_SIGNED_CSR	The key server has modified and signed the certificate. It is provided back and this function stores now the valid certificate in the configured storage.
KEYM_SERVICE_CERT_SET_ROOT	The key server requests to store a root certificate. The service checks if the certificate slot is empty and if so will validate the root certificate according to the configured rule and will store the root certificate
KEYM_SERVICE_CERT_UPDATE_ROOT	The key server requests to update an existing root certificate. The service checks if a root certificate exists and verifies the new root certificate against the already existing ones. If the validation was successful, the root certificate is re-newed in the slot.
KEYM_SERVICE_CERT_SET_INTERMEDIATE	The key server requests to store an intermediate certificate. A root certificate shall already exist to allow to validate the intermediate certificate against the root certificate and other certificates that might exist in the chain. The certificate slot is checked to be empty. If the validation was successful, the certificate is stored in the slot.
KEYM_SERVICE_CERT_UPDATE_INTERMEDIATE	The key server requests to update an intermediate certificate. It is verified against the root certificate and other certificates that might exist in the chain. If the validation was successful the certificate is updated.
KEYM_SERVICE_CERT_UPDATE_CRL	The key server provides a certificate revocation list. The service checks the signature of the list and stores it in the slot if the validation was successful. The revocation list shall then be checked during the verification of certificates if at least one CRL is available.

The implementation of either or all of the services are optional.
⌋(SRS_CryptoStack_00023, SRS_CryptoStack_00111, SRS_CryptoStack_00061)

[SWS_KeyM_00114] ⌈ If *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to `TRUE`, then a key update session shall be started before a service certificate operation can be performed.
⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00149] ⌈ The service operation runs asynchronously and will call `KeyM_ServiceCertificateCallbackNotification()` with results when the operation has finished.
⌋(SRS_CryptoStack_00106, SRS_CryptoStack_00101)

8.3.3.2 KeyM_SetCertificate

[SWS_KeyM_00057]⌈

Service Name	KeyM_SetCertificate	
Syntax	`Std_ReturnType KeyM_SetCertificate ( KeyM_CertificateIdType CertId, const KeyM_CertDataType* CertificateDataPtr )`	
Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CertId	Holds the identifier of the certificate
	CertificateDataPtr	Pointer to a structure that provides the certificate data.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	E_OK: Certificate accepted. E_NOT_OK: Certificate could not be set. KEYM_E_PARAMETER_MISMATCH: Parameter do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match
Description	This function provides the certificate data to the key management module to temporarily store the certificate.	
Available via	KeyM.h	

⌋()

The `KeyM_SetCertificate()` function is used to store a given certificate to verify it against a certificate chain. Certificates from the chain can either be provided temporarily in dedicated certificate slots and stored with `KeyM_SetCertificate()` or are permanently stored with the `KeyM_ServiceCertificate()`. This can be done, for example, through proprietary operations during the manufacturing process. At least it is necessary for a proper operation, that the root certificate is available.

[SWS_KeyM_00115] ⌈ If all parameters are accepted the function shall store the provided certificate data in an internal memory that is assigned to the certificate slot

referenced by the given CertId, typically in RAM. Once the certificate is provided the certificate submodule will start parsing the certificate.
⌋(SRS_CryptoStack_00111)

The parsing of a certificate can either be done directly within this function or can be operated in the background or main function.

Note: Setting the certificate and parsing it successfully does not necessarily imply that the certificate is validated in its chain of trust. The parsing is merely a pre-requisite to perform a certificate validation which is requested with another function.

[SWS_KeyM_00116] ⌈ The function returns `E_OK` if the certificate was basically accepted. Any other return value indicates that the certificate was not accepted. No parsing and validation operation can be performed on this certificate until a new certificate is provided and accepted.
⌋(SRS_CryptoStack_00096)

Info: The status of the certificate if it is parsed or validated successfully can be checked with `KeyM_CertGetStatus()`.

[SWS_KeyM_00166] ⌈
If the storage class of the certificate referenced by the container *KeyMCertificate*/*KeyMCertificateStorage* is set to `KEYM_STORAGE_IN_CSM` or `KEYM_STORAGE_IN_NVM` a development error `KEYM_E_CONFIG_FAILURE` shall be generated. If development mode is disabled the value `E_NOT_OK` shall be returned.
⌋(SRS_CryptoStack_00118, SRS_CryptoStack_00096)

[SWS_KeyM_00141] ⌈ The status of a certificate can be reset by calling `KeyM_SetCertificate()` with the corresponding certificate ID but with length information 0. The function will return `E_OK` and will reset the status of the certificate to `KEYM_CERTIFICATE_NOT_AVAILABLE` (see `KeyM_CertGetStatus()`).
⌋(SRS_CryptoStack_00096, SRS_CryptoStack_00115)

8.3.3.3 KeyM_GetCertificate

[SWS_KeyM_00058]⌈

Service Name	KeyM_GetCertificate

Syntax	`Std_ReturnType KeyM_GetCertificate (`
	`  KeyM_CertificateIdType CertId,`
	`  KeyM_CertDataType* CertificateDataPtr`
	`)`

Service ID [hex]	0x0b
Sync/Async	Synchronous

Reentrancy	Non Reentrant	
Parameters (in)	CertId	Holds the identifier of the certificate
Parameters (inout)	Certificate DataPtr	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.
Parameters (out)	None	
Return value	Std_-Return-Type	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.
Description	This function provides the certificate data	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00117] ⌈ This function shall provide certificate data referenced by certificate ID. It retrieves the information from the corresponding slot, checks if the data structure references a data buffer that is large enough to store the requested certificate, copies the data into the elements of CertificateDataPtr and adjusts the size. The function returns `E_OK` on success, or any other appropriate return value if the certificate data cannot be provided.
⌋(SRS_CryptoStack_00112, SRS_CryptoStack_00096)

8.3.3.4 KeyM_VerifyCertificates

[SWS_KeyM_00059]⌈

Service Name	KeyM_VerifyCertificates

Syntax	`Std_ReturnType KeyM_VerifyCertificates (`
	`  KeyM_CertificateIdType CertId,`

	`KeyM_CertificateIdType CertUpperId )`	
Service ID [hex]	0x0c	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CertId	Holds the identifier of the lower certificate in the chain
	CertUpperId	Holds the identifier of the upper certificate in the chain
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	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. 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.
Description	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.	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00118] ⌈ The function shall validate two certificates referenced by certificate IDs. Both certificate data shall be present, the certificate referenced by CertUpperId shall have been validated before, otherwise the function will return `KEYM_E_CERT_INVALID_CHAIN_OF_TRUST`.
⌋(SRS_CryptoStack_00111)

[SWS_KeyM_00119] ⌈ The function returns `E_OK` if the validation request was accepted. Any other return value indicates an error and the validation will not be

started. It does not perform the validation operation directly, but in the background. A callback will be called after validation to provide the result.
⌋(SRS_CryptoStack_00106, SRS_CryptoStack_00096)

[SWS_KeyM_00123] ⌈ After the certificate submodule has successfully validated the certificate, the corresponding public key shall be stored in the assigned key element of the CSM. This allows the application to operate jobs where this key is assigned to.
⌋(SRS_CryptoStack_00118)

[SWS_KeyM_00139] ⌈ If a certificate shall be verified but has not yet been parsed, the parsing operation shall be done as soon as possible and the verification process shall be started afterwards.
⌋(SRS_CryptoStack_00111, SRS_CryptoStack_00031)

8.3.3.5 KeyM_VerifyCertificate

[SWS_KeyM_00060]⌈

Service Name	KeyM_VerifyCertificate	

Syntax	`Std_ReturnType KeyM_VerifyCertificate (`
	`  KeyM_CertificateIdType CertId`
	`)`	

Service ID [hex]	0x0d	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CertId	Holds the identifier of the certificate
Parameters (inout)	None	
Parameters (out)	None	
Return value	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. 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.
Description	This function verifies a certificate that was previously provided with KeyM_Set Certificate() against already stored and provided certificates stored with other certificate IDs.	
Available via	KeyM.h	

⌋()

The intention of `KeyM_VerifyCertificate()` is to autonomously identify the certificates referenced by CertID and the associated certificates in the chain. The certificate that shall be validated is expected to be set prior to this function call with `KeyM_SetCertificate()`. If a certificate in the chain is not yet verified, it will be parsed and verified automatically until the complete chain of trust has been parsed and verified up to the root certificate. The verification shall be done from the top of the certificate hierarchy to the bottom. Thus, the function shall first identify the chain of trust and check if the root certificate has been validated. If this is valid, the next intermediate certificate shall be checked until the certificate referenced by CertID is to be verified. The order of the validation is important to meet security requirements.

[SWS_KeyM_00120] ⌈ The verification of the certificate(s) shall be done asynchronously. All certificates that are involved in the chain of trust shall be verified, from top to bottom. The callback function `KeyM_CertificateVerifyCallbackNotification()` shall be called if the verification has been finished and provide the result of the operation in the callback.
⌋(SRS_CryptoStack_00106, SRS_CryptoStack_00101)

[SWS_KeyM_00121] ⌈ The function returns `E_OK` if the operation has been accepted and can be performed. Any other return value will indicate the appropriate error and the verification will not be started.
⌋(SRS_CryptoStack_00096)

[SWS_KeyM_00135] ⌈ Elements of the certificate associated and defined in *KeyMCertificateElement* and subcontainers shall be used to verify elements of the certificate according to the configuration. This shall be done for every certificate that has to be verified.
⌋(SRS_CryptoStack_00031, SRS_CryptoStack_00111)

8.3.3.6 KeyM_VerifyCertificateChain

[SWS_KeyM_00061]⌈

Service Name	KeyM_VerifyCertificateChain

Syntax	`Std_ReturnType KeyM_VerifyCertificateChain (`
	`  KeyM_CertificateIdType CertId,`

	`const KeyM_CertDataType[] certChainData,`
	`uint8 NumberOfCertificates`
	`)`	

Service ID [hex]	0x0e	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	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.
	NumberOf Certificates	Defines the number of certificates stored in the CertChainData array.
Parameters (inout)	None	
Parameters (out)	None	
Return value	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. 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.
Description	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 Key M_SetCertificate() and that the root certificate is either in the list or is already assigned to one of the other certificates.	
Available via	KeyM.h	

⌋()

The function `KeyM_VerifyCertificateChain()` is called when a certificate shall be validated, but there are one or more other certificates that is required for the chain of trust. For example, a PKI consists of four certificates, including the root certificate and the certificate used for authentication. Two other certificates are not permanently available in the configuration and they are just needed to proof the authentication of the one in place. Thus, only the to-be-verified certificate need to be set with `KeyM_SetCertificate()` while the other two certificates of the chain can be provided in a temporary buffer. They are needed to complete the chain of trust. The verification will start by identifying the permanently provided certificate, namely the root certificate in-place. This certificate is checked followed by any other permanently stored certificates until the missing one in the chain. These certificates are referenced by certChainData. The first one from the list will be parsed and verified against the last one that has been permanently stored in the certificate submodule. This would be the root certificate in our example. If the first certificate in certChainData can be verified against the root certificate, the next one in certChainData will be verified against the previously verified until all certificates in certChainData have been verified. The last one in the list will then be used to verify the certificate referenced with CertId. Only the final result of this verification is important and need to be stored. The intermediate results for the verification of certChainData is not important and can be dropped.

[SWS_KeyM_00124] ⌈ The verification of the certificate(s) shall be done asynchronously. All certificates that are involved in the chain of trust shall be verified, from top to bottom. The callback function `KeyM_CertificateVerifyCallbackNotification()` shall be called if the verification has been finished and provide the result of the operation in the callback.
⌋(SRS_CryptoStack_00106, SRS_CryptoStack_00101)

[SWS_KeyM_00125] ⌈ The function returns `E_OK` if the operation has been accepted and can be performed. Any other return value will indicate the appropriate error and the verification will not be started.
⌋(SRS_CryptoStack_00096)

[SWS_KeyM_00126] ⌈ After the certificate submodule has successfully validated the certificate, the corresponding public key shall be stored in the assigned key element of the CSM. This allows the application to operate jobs where this key is assigned to. This has to be done each time a verification of a certificate was successfully performed, regardless of the function call that has been used.
⌋(SRS_CryptoStack_00118)

8.3.3.7 KeyM_CertElementGet

[SWS_KeyM_00063]⌈

Service Name	KeyM_CertElementGet

Syntax	`Std_ReturnType KeyM_CertElementGet (`
	`  KeyM_CertificateIdType CertId,`
	`  KeyM_CertElementIdType CertElementId,`
	`  uint8* CertElementData,`
	`  uint32* CertElementDataLength`
	`)`

Service ID [hex]	0x0f	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CertId	Holds the identifier of the certificate.
	Cert ElementId	Specifies the ElementId where the data shall be read from.
Parameters (inout)	Cert Element DataLength	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.
Parameters (out)	Cert Element Data	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 value	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.
Description	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.	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00127] ⌈ The function shall retrieve certificate elements from the certificate as defined in the configuration by searching the object ID in the configured section of the certificate and provide the data from the parsed and validated certificate by copying the content into the provided data buffer when the indicated buffer size is large enough.
⌋(SRS_CryptoStack_00112)

8.3.3.8 KeyM_CertElementGetByIndex

[SWS_KeyM_91014]⌈

Service Name	KeyM_CertificateElementGetByIndex	

Syntax	`Std_ReturnType KeyM_CertificateElementGetByIndex (`
	`  KeyM_CertificateIdType CertId,`
	`  KeyM_CertElementIdType CertElementId,`
	`  uint32 Index,`
	`  uint8* CertElementDataPtr,`
	`  uint32* CertElementDataLengthPtr`
	`)`	

Service ID [hex]	0x1b	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CertId	Identifier of the certificate where the element shall be read from.
	CertElementId	Specifies the ElementId where the data shall be read from.
	Index	Specifies the index to the element that shall be read (0..N).
Parameters (inout)	CertElementDataLengthPtr	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.
Parameters (out)	CertElementDataPtr	Pointer to a data buffer allocated by the caller of this function. If the function returns E_OK element data are copied into this buffer.
Return value	Std_ReturnType	E_OK: Element found and data provided in the buffer. E_NOT_OK: Unable to read the element data. KEYM_E_PARAMETER_MISMATCH: Invalid certificate ID, element ID invalid or index out of range. 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 is empty. KEYM_E_CERT_INVALID: Certificate is not valid or not verified successfully
Description	This function provides the element data of a certificate. The function is used if an element type can have more than one parameter. The index specifies which element shall be read. The function works similar to the KeyM_CertElementGetFirst/KeyM_	

	CertElementGetNext, but instead of the iteration, the individual elements can be accessed by index (like the operation in the service interface)
Available via	KeyM.h

⌋()

8.3.3.9 KeyM_CertElementGetCount

[SWS_KeyM_91015]⌈

Service Name	KeyM_CertificateElementGetCount	

Syntax	`Std_ReturnType KeyM_CertificateElementGetCount (`
	`  KeyM_CertificateIdType CertId,`
	`  KeyM_CertElementIdType CertElementId,`
	`  uint16* CountPtr`
	`)`	

Service ID [hex]	0x1c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CertId	Identifier of the certificate.
	CertElementId	Specifies the certificate element.
Parameters (inout)	None	
Parameters (out)	CountPtr	Pointer to the buffer where the number of available data elements for this certificate element shall be copied to.
Return value	Std_Return-Type	E_OK: Count value has been provided. E_NOT_OK: Unable to provide the count value. KEYM_E_PARAMETER_MISMATCH: Certificate ID or certificate element ID invalid resp. out of range.
Description	This function provides the total number of data elements that are available for the specified certificate element. Typically, only one data element is available. But in some cases, several data elements can be assigned to one certificate element in a row. This function retrieves the total number of elements. The individual data elements can then accessed with KeyM_CertificateElementGetByIndex(). It is similar to the functions KeyM_CertElementGetFirst/KeyM_CertElementGetNext to retrieve a group of data elements of one certificate element.	

Available via	KeyM.h

⌋()

8.3.3.10 KeyM_CertElementGetFirst

[SWS_KeyM_00064]⌈

Service Name	KeyM_CertElementGetFirst	

Syntax	`Std_ReturnType KeyM_CertElementGetFirst (`
	`  KeyM_CertificateIdType CertId,`
	`  KeyM_CertElementIdType CertElementId,`
	`  KeyM_CertElementIteratorType* CertElementIterator,`
	`  uint8* CertElementData,`
	`  uint32* CertElementDataLength`
	`)`	

Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Reentrant Reentrant for one iterator.	
Parameters (in)	CertId	Holds the identifier of the certificate.
	CertElement Id	Specifies the CertElementId where the data shall be read from.
Parameters (inout)	CertElement Iterator	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().
	CertElement DataLength	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.
Parameters (out)	CertElement Data	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 value	Std_Return-Type	E_OK: Element found and data provided in the buffer. The cert ElementIterator has been initialized accordingly. E_NOT_OK: Element data not found. CertElementIterator cannot be used for further calls. 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 is empty. KEYM_E_CERT_INVALID: Certificate is not valid or not verified successfully
Description	This function is used to initialize the interative 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().	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00128] ⌈ The function shall retrieve certificate elements from the certificate as defined in the configuration by searching the object ID in the configured section of the certificate. If no error is detected, the identified data from the parsed and validated shall be provided from the certificate by copying the content into the provided data buffer when the indicated buffer size is large enough and the function shall return `E_OK`. Otherwise, any other appropriate error code shall be provided.
⌋(SRS_CryptoStack_00096, SRS_CryptoStack_00112)

[SWS_KeyM_00129] ⌈ The function returns `E_OK`, the iterator structure shall be initialized in a way, that further listed elements associated to the referenced certificate element can be retrieved one after another.
⌋(SRS_CryptoStack_00112)

Rationale:
Some certificate elements can contain more than one element associated to an object ID. The function pair of `KeyM_CertElementGetFirst()`/ `KeyM_CertElementGetNext()` shall be used to retrieve a list of elements one after another. The iterator, which is implementation specific, shall be used to forward iterate through the list of elements.

8.3.3.11 KeyM_CertElementGetNext

[SWS_KeyM_00065]⌈

Service Name	KeyM_CertElementGetNext

Syntax	`Std_ReturnType KeyM_CertElementGetNext (`
	`  KeyM_CertElementIteratorType* CertElementIterator,`
	`  uint8* CertElementData,`

	`uint32* CertElementDataLength )`	
Service ID [hex]	0x11	
Sync/Async	Synchronous	
Reentrancy	Reentrant Reentrant for one iterator.	
Parameters (in)	None	
Parameters (inout)	CertElement Iterator	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.
	CertElement DataLength	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.
Parameters (out)	CertElement Data	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 value	Std_Return- Type	E_OK: Element found and data provided in the buffer. The Cert ElementIterator has been initialized accordingly. E_NOT_OK: Element data not found. CertElementIterator cannot be used for further calls. 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 is empty. KEYM_E_CERT_INVALID: Certificate is not valid or not verified successfully
Description	This function provides further data from a certificate element, e.g. if a set of data are 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.	
Available via	KeyM.h	

⌋()

[SWS_KeyM_00148] ⌈ This function can only be called for certificate elements where *KeyMCertificateElementHasIteration* is set to `TRUE`. Otherwise, the function shall return `KEYM_E_CERT_INVALID`.
⌋(SRS_CryptoStack_00112)

[SWS_KeyM_00130] ⌈ The function `KeyM_CertGetElementFirst()` shall be called once with return value `E_OK` before the `KeyM_CertGetElementNext()` can be called.
⌋(SRS_CryptoStack_00112)

[SWS_KeyM_00131] ⌈ If `KeyM_CertGetElementNext()` returns any other value than `E_OK`, no further function call to `KeyM_CertElementGetNext()` is allowed with the iterator structure until a new a successful call to `KeyM_CertElementGetFirst()` was performed.
⌋(SRS_CryptoStack_00112)

[SWS_KeyM_00132] ⌈ The function `KeyM_CertGetElementNext()` returns `E_OK` and provides further data from the list referenced by certificate and certificate element ID used by the call to `KeyM_CertGetElementFirst()`.
⌋(SRS_CryptoStack_00096, SRS_CryptoStack_00112)

8.3.3.12 KeyM_CertGetStatus

[SWS_KeyM_00066]⌈

Service Name	KeyM_CertGetStatus	

Syntax	`Std_ReturnType KeyM_CertGetStatus (`
	`  KeyM_CertificateIdType CertId,`
	`  KeyM_CertificateStatusType* Status`
	`)`	

Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CertId	Holds the identifier of the certificate
Parameters (inout)	None	
Parameters (out)	Status	Provides the status of the certificate.
Return value	Std_ReturnType	E_OK Status successfully provided E_NOT_OK Status provision currently not possible. KEYM_E_PARAMETER_MISMATCH: Invalid certificate ID.

Description	This function provides the status of a certificate.
Available via	KeyM.h

⌋()

[SWS_KeyM_00133] ⌈ The certificate submodule shall maintain the status of a certificate and provide the status on demand.
⌋(SRS_CryptoStack_00115)
[SWS_KeyM_00134] ⌈ A certificate has the status `KEYM_CERTIFICATE_VALID` if it was parsed and verified completely against other certificates of the PKI. All certificates of the chain of trust are available and verified completely.
⌋(SRS_CryptoStack_00031, SRS_CryptoStack_00115)

[SWS_KeyM_00136] ⌈ A certificate is in the status `KEYM_CERTIFICATE_INVALID` if the contents could not be parsed due to an internal error, e.g. a format error, signature failure period failure or any other failure occurred during the verification.
⌋(SRS_CryptoStack_00115, SRS_CryptoStack_00023)

[SWS_KeyM_00137] ⌈ A certificate has the status `KEYM_CERTIFICATE_PARSED_NOT_VALID` if the certificate has been provided e.g. with the function `KeyM_SetCertificate()` and has been parsed successfully, but the verification has not yet been initiated, e.g. by a call to `KeyM_VerifyCertificate()`.
⌋(SRS_CryptoStack_00115))

[SWS_KeyM_00138] ⌈ A certificate has the status `KEYM_CERTIFICATE_NOT_PARSED` if the certificate was already provided, e.g. with `KeyM_SetCertificate()` but the parsing process is still ongoing in the background.
⌋(SRS_CryptoStack_00115))

[SWS_KeyM_00140] ⌈ A certificate is in the status `KEYM_CERTIFICATE_NOT_AVAILABLE` if the certificate has not yet been provided by a function call `KeyM_SetCertificate()` or the function was called with the certificate ID but with certificate length of 0.
⌋(SRS_CryptoStack_00115))

8.4 Call-out definitions

The KeyM module provides no callouts.

8.5 Scheduled functions

8.5.1 KeyM_MainFunction

[SWS_KeyM_00074]⌈

Service Name	KeyM_MainFunction

Syntax	`void KeyM_MainFunction (`
	`  void`
	`)`

Service ID [hex]	0x19
Description	Function is called periodically according the specified time interval.
Available via	SchM_KeyM.h

⌋()

8.5.2 KeyM_MainBackgroudFunction

[SWS_KeyM_00075]⌈

Service Name	KeyM_MainBackgroundFunction

Syntax	`void KeyM_MainBackgroundFunction (`
	`  void`
	`)`

Service ID [hex]	0x1a
Description	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 such as KeyM_KH_Update().
Available via	SchM_KeyM.h

⌋()

8.6 Expected Interfaces

In this chapter all external interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This chapter defines all external interfaces which are required to fulfill the core functionality of the module.

[SWS_KeyM_00076]⌈

API Function	***Header File***	***Description***
Csm_Key-ElementGet	Csm.h	Retrieves the key element bytes from a specific key element of the key identified by the keyId and stores the key element in the memory location pointed by the key pointer.
Csm_Key-ElementSet	Csm.h	Sets the given key element bytes to the key identified by keyId.
Csm_KeySet-Valid	Csm.h	Sets the key state of the key identified by keyId to valid.

⌋()

8.6.2 Optional Interfaces

This chapter defines all external interfaces which are required to fulfill an optional functionality of the module.

[SWS_KeyM_00078]⌈

API Function	***Header File***	***Description***
Csm_KeyDerive	Csm.h	Derives a new key by using the key elements in the given key identified by the keyId. The given key contains the key elements for the password and salt. The derived key is stored in the key element with the id 1 of the key identified by targetCryptoKeyId.
Csm_Signature-Verify	Csm.h	Verifies the given MAC by comparing if the signature is generated with the given data.
Det_ReportError	Det.h	Service to report development errors.
IdsM_Set-SecurityEvent	IdsM.h	This API is the application interface to report security events to the Ids M.

IdsM_Set-SecurityEvent-WithContextData	IdsM.h	This API is the application interface to report security events with context data to the IdsM.
StbM_Get-CurrentTime	StbM.h	Returns a time value (Local Time Base derived from Global Time Base) in standard format. Note: This API shall be called with locked interrupts / within an Exclusive Area to prevent interruption (i.e., the risk that the time stamp is outdated on return of the function call).

⌋()

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The names of these kind of interfaces is not fixed because they are configurable.

Hint:
The functional behaviour of key handler functions is described in the respective section of the calling Key Management function.

8.6.3.1 KeyM_KH_Start

[SWS_KeyM_00067]⌈

Service Name	KeyM_KH_Start	
Syntax	`Std_ReturnType KeyM_KH_Start ( KeyM_StartType StartType, const uint8* RequestData, uint16 RequestDataLength, uint8* ResponseData, uint16* ResponseDataLength )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	StartType	Defines in which mode the key operation shall be executed.
	RequestData	Information that comes along with the request, e.g. signature

	RequestData Length	Length of data in the RequestData array
Parameters (inout)	ResponseData Length	In: Max number of bytes available in ResponseData Out: Actual number of bytes in ResponseData if function returns E_OK.
Parameters (out)	ResponseData	Data returned by the function.
Return value	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 do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match
Description	If KeyMCryptoKeyStartFinalizeFunctionEnabled and KeyMCryptoKeyHandlerStart FinalizeEnabled is set to TRUE, this function will be called immediately when KeyM_ Start gets called. The function shall return E_OK to switch the Key Manager into the active state for any key operation.	
Available via	KeyM_Externals.h	

⌋()

8.6.3.2 KeyM_KH_Prepare

[SWS_KeyM_00068]⌈

Service Name	KeyM_KH_Prepare	

Syntax	`Std_ReturnType KeyM_KH_Prepare (`
	`  const uint8* RequestData,`
	`  uint16 RequestDataLength,`
	`  uint8* ResponseDataPtr,`
	`  uint16* ResponseDataLength`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	RequestData	Information that comes along with the request
	RequestData Length	Length of data in the RequestData array

Parameters (inout)	ResponseData Length	In: Max number of bytes available in ResponseData Out: Actual number of bytes in ResponseData.
Parameters (out)	ResponseDataPtr	Data returned by the function.
Return value	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 do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match
Description	If the configuration parameters KeyMCryptoKeyPrepareFunctionEnabled and Key MCryptoKeyHandlerPrepareEnabled are both set to TRUE, then this function will be called immediately when KeyM_Prepare gets called. The function takes over the task to prepare a key management operation. The response data will be passed on as is to the caller of Key_Prepare.	
Available via	KeyM_Externals.h	

⌋()

8.6.3.3 KeyM_KH_Update

[SWS_KeyM_00069]⌈

Service Name	KeyM_KH_Update	

Syntax	`Std_ReturnType KeyM_KH_Update (`
	`  const uint8* KeyNamePtr,`
	`  uint16 KeyNameLength,`
	`  const uint8* RequestData,`
	`  uint16 RequestDataLength,`
	`  uint8* ResultDataPtr,`
	`  uint16* ResultDataLengthPtr,`
	`  KeyM_CryptoKeyIdType* KeymId,`
	`  KeyM_KH_UpdateOperationType* UpdateOperation`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	KeyName Ptr	Points to an array that defines the name of the key to be updated

	KeyName Length	Specifies the number of bytes in KeyNamePtr. The value 0 indicates that no KeyNamePtr is provided within this function.
	Request Data	Information that comes along with the request
	Request DataLength	Length of data in the RequestData array
Parameters (inout)	ResultData LengthPtr	In: Max number of bytes available in ResultDataPtr Out: Actual number of bytes in ResultData or 0 if no data available. Unspecified or untouched if return value indicates a failure.
	KeymId	Provides a reference to the crypto key as an index to the crypto key table. In: Providing the key ID if a name was provided and a key was found. Returns 0xFFFFFFFFul if no key was found. Out: Key ID of the key where the operation shall be performed to if updateOperation indicates a key operation.
Parameters (out)	ResultData Ptr	Data returned by the function.
	Update Operation	Provides information to the caller what operation has been performed and how to interpret the ResultData.
Return value	Std_Return-Type	E_OK: Data returned by this function. E_NOT_OK: General error, no data provided. E_BUSY: Service could not be accepted because another operation is already ongoing. Try next time. KEYM_E_PARAMETER_MISMATCH: A parameter does not have expected value. Service discarded. KEYM_E_KEY_CERT_WRITE_FAIL: Key could not be written. KEYM_E_KEY_CERT_UPDATE_FAIL: General failure on updating a key.
Description	If the configuration item KeyMCryptoKeyHandlerUpdateEnabled is set to TRUE, the KeyM_Update function will not perform any operation but will delegate the operation to the key handler. On return, the function provides the status of the key operation.	
Available via	KeyM_Externals.h	

⌋()

[SWS_KeyM_00097] ⌈ If a key handler is used for key update operation (*KeyMCryptoKeyHandlerUpdateEnabled* is set to `TRUE`), the Key Manager shall provide a pointer to an internal buffer to the key handler when calling `KeyM_KH_Update()`. This buffer can be used by the key handler to store the key data results during the operation. As a consequence, the `KeyM_Update()` function shall not touch this buffer after calling `KeyM_KH_Update()` until the key handler

returns. The length of the buffer shall be at least as large as the largest value of all *KeyMCryptoKey/KeyMCryptoKeyMaxLength* defined in the KeyMCryptoKey container.
⌋(SRS_CryptoStack_00107)

[SWS_KeyM_00096] ⌈ If the key handler returns `E_OK` and provides the operation type `KEYM_KH_UPDATE_KEY_UPDATE_REPEAT` and ResultDataLengthPtr indicates a value greater than 0 then the key manager shall perform the key update operation according to the configuration (store or derive the key in CSM) and use the data stored in ResultDataPtr.
If the update operation was successful, the key handler shall be called again.
⌋(SRS_CryptoStack_00107, SRS_CryptoStack_00109)

Info: The repeated call to the key handler update operation allows the key handler to update several keys at a time.

[SWS_KeyM_00093] ⌈ If the key handler returns and provides the operation type `KEYM_KH_UPDATE_FINISH`, the key update operation shall finish and use the return value from the key handler. The data buffer from `KeyM_KH_Update`::ResultDataPtr shall be copied to the buffer provided with `KeyM_Update`::ResultDataPtr and the `KeyM_CryptoKeyUpdateCallbackNotification()` function shall be called by the `KeyM_Update()` function.
⌋ (SRS_CryptoStack_00107)

Info:
This allows the key handler update operation to provide results back to the key server.

8.6.3.4 KeyM_KH_Finalize

[SWS_KeyM_00070]⌈

Service Name	KeyM_KH_Finalize	

Syntax	`Std_ReturnType KeyM_KH_Finalize (`
	`  const uint8* RequestData,`
	`  uint16 RequestDataLength,`
	`  uint8* ResponseData,`
	`  uint16* ResponseDataLength`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
	RequestData	Information that comes along with the request

Parameters (in)	RequestData Length	Length of data in the RequestData array
Parameters (inout)	ResponseData Length	In: Max number of bytes available in ResponseData Out: Actual number of bytes in ResponseData.
Parameters (out)	ResponseData	Data returned by the function.
Return value	Std_ReturnType	E_OK: Operation has been accepted and will be processed internally. Results will be provided 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. KEYM_E_PARAMETER_MISMATCH: Parameter do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match
Description	If KeyMCryptoKeyStartFinalizeFunctionEnabled and KeyMCryptoKeyHandlerStart FinalizeEnabled is set to TRUE, this function will be called immediately when KeyM_ Finalize gets called KeyM_Finalize() will not perform any operation but will call this key handler function to delegate the operation.	
Available via	KeyM_Externals.h	

⌋()

8.6.3.5 KeyM_KH_Verify

[SWS_KeyM_00071]⌈

Service Name	KeyM_KH_Verify	

Syntax	`Std_ReturnType KeyM_KH_Verify (`
	`  const uint8* KeyNamePtr,`
	`  uint16 KeyNameLength,`
	`  const uint8* RequestData,`
	`  uint16 RequestDataLength,`
	`  uint8* ResponseData,`
	`  uint16* ResponseDataLength`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
	KeyNamePtr	Pointer to an array that defines the name of the key to be updated

Parameters (in)	KeyName Length	Specifies the number of bytes in keyName. The value 0 indicates that no keyName is provided within this function.
	RequestData	Information that comes along with the request
	RequestData Length	Length of data in the RequestData array
Parameters (inout)	ResponseData Length	In: Max number of bytes available in ResponseData Out: Actual number of bytes in ResponseData.
Parameters (out)	ResponseData	Data returned by the function.
Return value	Std_Return-Type	KEYM_E_PENDING: Operation runs in asynchronous mode, has been accepted and will be processed internally. Results will be provided through callback E_OK: Operation was successfully performed. Result information are available. 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 (for asynchronous mode). KEYM_E_PARAMETER_MISMATCH: Parameter do 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.
Description	If KeyMCryptoKeyHandlerVerifyEnabled is set to TRUE, the KeyM_Verify function will not perform any operation but will delegate its operation to this service callback. The intention is to perform a verification of input data using the CSM job referenced with KeyMCryptoKeyCsmVerifyJobRef.	
Available via	KeyM_Externals.h	

⌋()

8.6.3.6 KeyM_KH_ServiceCertificate

[SWS_KeyM_00072]⌈

Service Name	KeyM_KH_ServiceCertificate

Syntax	`Std_ReturnType KeyM_KH_ServiceCertificate (`
	`  KeyM_ServiceCertificateType Service,`
	`  const uint8* CertName,`
	`  uint16 CertNameLength,`

	`const uint8* RequestData,`
	`uint16 RequestDataLength,`
	`uint8* ResponseData,`
	`uint16* ResponseDataLength`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Service	Provides the type of service the certificate submodule has to perform.
	CertName	Points to an array that defines the name of the key to be updated
	CertNameLength	Specifies the number of bytes in keyName. The value 0 indicates that no keyName is provided within this function.
	RequestData	Information that comes along with the request
	RequestData Length	Length of data in the RequestData array
Parameters (inout)	ResponseData Length	In: Max number of bytes available in ResponseData Out: Actual number of bytes in ResponseData.
Parameters (out)	ResponseData	Data returned by the function.
Return value	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 do not match with expected value. KEYM_E_KEY_CERT_SIZE_MISMATCH: Parameter size doesn't match
Description	If KeyMCryptoKeyHandlerServiceCertificateEnabled is set to TRUE, this function will be called by KeyM_ServiceCertificate() to delegate the operation to this user specific service function.	
Available via	KeyM_Externals.h	

⌋()

8.6.3.7 KeyM_CryptoKeyUpdateCallbackNotification

[SWS_KeyM_00077]⌈

Service Name	KeyM_CryptoKeyUpdateCallbackNotification	
Syntax	`void KeyM_CryptoKeyUpdateCallbackNotification ( KeyM_ResultType Result, uint16 ResultDataLength, const uint8* ResultDataPtr )`	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	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.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notifies the application that a crypto key update operation has been finished. This function is used by the key manager.	
Available via	KeyM_Externals.h	

⌋()

[SWS_KeyM_00150] ⌈ This callback function indicates the end of a key update operation. It is called after a successful call to `KeyM_Update()` that has returned `E_OK` and the requested key update operation was finished. It is only needed if *KeyMCryptoKeyManagerEnabled* is set to `TRUE`.

⌋(SRS_CryptoStack_00106)

8.6.3.8 KeyM_CryptoKeyFinalizeCallbackNotification

[SWS_KeyM_00079]⌈

Service Name	KeyM_CryptoKeyFinalizeCallbackNotification	
Syntax	`void KeyM_CryptoKeyFinalizeCallbackNotification ( KeyM_ResultType Result, uint16 ResultDataLength, const uint8* ResultDataPtr )`	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Result	Contains information about the result of the operation.
	ResultData Length	ontains the length of the resulting data of this operation.
	ResultDataPtr	Pointer to the data of the result (the data buffer that has been provided with the service function).
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notifies the application that a crypto key finalize operation has been finished. The callback function is only called and needed if KeyMCryptoKeyStartFinalizeFunction Enabled is set to TRUE.	
Available via	KeyM_Externals.h	

⌋()

[SWS_KeyM_00080] ⌈ If *KeyMCryptoKeyStartFinalizeFunctionEnabled* is set to `TRUE` the callback function `KeyM_CryptoKeyFinalizeCallbackNotification()` indicates that the finalize operation has been concluded. The result value provides the status of the finalization operation, if all keys have been validated successfully or not. The ResultData can provide additional information about the finalization operation used to provide this back to the key server.
⌋ (SRS_CryptoStack_00106)

8.6.3.9 KeyM_CryptoKeyVerifyCallbackNotification

[SWS_KeyM_00081]⌈

Service Name	KeyM_CryptoKeyVerifyCallbackNotification	
Syntax	`void KeyM_CryptoKeyVerifyCallbackNotification ( KeyM_ResultType Result, uint32 KeyId, uint16 ResultDataLength, const uint8* ResultDataPtr )`	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Result	Contains information about the result of the operation.
	KeyId	The key identifier where this verification was started for.
	ResultDataLength	Contains the length of the resulting data of this operation if any.
	ResultDataPtr	Pointer to the data of the result.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notifies the application that a crypto key verify operation has been finished. This function is used by the key manager.	
Available via	KeyM_Externals.h	

⌋()

[SWS_KeyM_00151] ⌈ If *KeyMCryptoKeyVerifyFunctionEnabled* is set to `TRUE` and `KeyM_Verify()` has been called successfully and returned `E_OK` and if *KeyMCryptoKeyVerifyAsyncMode* is set to `TRUE` then the Key Manager will perform the verification operation in asynchronous mode. The function `KeyM_CryptoKeyVerifyCallbackNotification()` will be called by the Key Manager after the verification for the given key and will provide the result.

⌋(SRS_CryptoStack_00106, SRS_CryptoStack_00101)

8.6.3.10 KeyM_ServiceCertificateCallbackNotification

[SWS_KeyM_00147]⌈

Service Name	KeyM_ServiceCertificateCallbackNotification	

Syntax	`void KeyM_ServiceCertificateCallbackNotification (`
	`  KeyM_CertificateIdType CertId,`
	`  KeyM_ResultType Result,`
	`  uint16 ResultDataLength,`
	`  const uint8* ResultDataPtr`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CertId	The certificate identifier where this service was started for.
	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.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	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 Key MServiceCertificateFunctionEnabled is set to TRUE. The function name is configurable by KeyMServiceCertificateCallbackNotificationFunc.	
Available via	KeyM_Externals.h	

⌋()

[SWS_KeyM_00152] ⌈ If *KeyMServiceCertificateFunctionEnabled* is set to `TRUE` and `KeyM_ServiceCertificate()` was called successfully by returning `E_OK` and *KeyMServiceCertificateCallbackNotificationFunc* is configured with a valid function name, this function will get called for the corresponding certificate to indicate the result of the requested operation.
⌋(SRS_CryptoStack_00106)

8.6.3.11 KeyM_CertificateVerifyCallbackNotification

[SWS_KeyM_00073]⌈

Service Name	KeyM_CertificateVerifyCallbackNotification	
Syntax	`Std_ReturnType KeyM_CertificateVerifyCallbackNotification ( KeyM_CertificateIdType CertId, KeyM_CertificateStatusType Result )`	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CertId	The certificate identifier that has been verified.
	Result	Contains information about the result of the operation.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK
Description	Notifies the application that a certificate verification has been finished. The function name is configurable by KeyMCertificateVerifyCallbackNotificationFunc.	
Available via	KeyM_Externals.h	

⌋()

[SWS_KeyM_00153] ⌈ If a certificate verification request was successfully submitted by `KeyM_VerifyCertificate()`, `KeyM_VerifyCertificates()` or `KeyM_VerifyCertificateChain()` by returning `E_OK` and *KeyMCertificateVerifyCallbackNotificationFunc* is configured with a valid function name, this function will get called for the corresponding certificate to indicate the result of the verification operation.

⌋(SRS_CryptoStack_00106)

8.7 Service Interfaces

This chapter is an add-on to the specification of the KeyM module. Whereas the other parts of the specification define the behavior and the C-interfaces of the corresponding basic software module, this chapter formally describes the corresponding AUTOSAR services for SWC generated by the RTE. The interfaces described here will be visible on the VFB and are used to generate the RTE between application and the KEYM module.

8.7.1 Scope of this Chapter

This chapter defines blueprints of the AUTOSAR Interfaces of the Key Manager Service (KeyM).

According to TPS_GST_00081 these blueprints are placed in `ARPackage /AUTOSAR/KeyM`.

8.7.2 Data Types

8.7.2.1 KeyM_StartType

[SWS_KeyM_00038]⌈

Name	KeyM_StartType		
Kind	Enumeration		
Range	KEYM_START_OEM_ PRODUCTIONMODE	0x01	Key operation starts in OEM production mode
	KEYM_START_ WORKSHOPMODE	0x02	Key operation starts in workshop mode
	reserved	0x80- 0x9F	The range from 0x80-0x9F is reserved for user specific extensions
Description	This type specifies in which mode the key operation will start. The OEM production mode provides higher privileges compared to workshop mode.		
Variation	--		

Available via	Rte_KeyM_Type.h

⌋()

8.7.2.2 KeyM_CertElementIdType

[SWS_KeyM_00300]⌈

Name	KeyM_CertElementIdType
Kind	Type
Derived from	uint16
Description	Certificate Element handle.
Variation	--
Available via	Rte_KeyM_Type.h

⌋()

8.7.2.3 KeyM_CertificateIdType

[SWS_KeyM_00301]⌈

Name	KeyM_CertificateIdType
Kind	Type
Derived from	uint16
Description	Certificate handle.
Variation	--
Available via	Rte_KeyM_Type.h

⌋()

8.7.2.4 KeyM_ServiceCertificateType

[SWS_KeyM_00039]⌈

Name	KeyM_ServiceCertificateType		
Kind	Enumeration		
Range	KEYM_SERVICE_CERT_ REQUEST_CSR	0x01	Key server requests to generate a certificate from the key client.
	KEYM_SERVICE_CERT_ UPDATE_SIGNED_CSR	0x02	Key server returns a previously received certificate and has been now signed by the CA.
	KEYM_SERVICE_CERT_ SET_ROOT	0x03	Key server wants to add a new root certificate.
	KEYM_SERVICE_CERT_ UPDATE_ROOT	0x04	Key server wants to update an existing root certificate.
	KEYM_SERVICE_CERT_ SET_INTERMEDIATE	0x05	Key server wants to add a new CA certificate. pre-requisite: Root certificate shall have been stored beforefor a successful verification.
	KEYM_SERVICE_CERT_ UPDATE_INTERMEDIATE	0x06	Key server wants to update an existing CA certificate.
	KEYM_SERVICE_CERT_ UPDATE_CRL	0x07	Provide or update a certificate revocation list.
	reserved	0x80-0x9F	The range from 0x80-0x9F is reserved for user specific extensions
Description	This type specifies the requested service operation and what information is provided with this function.		
Variation	--		
Available via	Rte_KeyM_Type.h		

⌋()

8.7.2.5 KeyM_KeyCertNameDataType

[SWS_KeyM_91000]⌈

Name	KeyM_KeyCertNameDataType

Kind	Array	***Element type***	uint8
Size	{ecuc(KeyM/KeyMGeneral/KeyMKeyCertNameMaxLength)} Elements		
Description	Array long enough to store the key or certificate name. baseTypeEncoding = UTF-8		
Variation	--		
Available via	Rte_KeyM_Type.h		

⌋()

8.7.2.6 KeyM_CertificateStatusType

[SWS_KeyM_91003]⌈

Name	KeyM_CertificateStatusType		
Kind	Enumeration		
Range	KEYM_CERTIFICATE_VALID	0x00	Certificate successfully parsed and verified.
	KEYM_CERTIFICATE_INVALID	0x01	The certificate is invalid (unspedified failure)
	KEYM_CERTIFICATE_NOT_PARSED	0x02	Certificate has not been parsed so far.
	KEYM_CERTIFICATE_PARSED_NOT_VALIDATED	0x03	Certificate parsed but not yet validated
	KEYM_CERTIFICATE_NOT_AVAILABLE	0x04	Certificate not set
	KEYM_E_CERTIFICATE_VALIDITY_PERIOD_FAIL	0x05	Certificate verification failed - Invalid Time Period
	KEYM_E_CERTIFICATE_SIGNATURE_FAIL	0x06	Certificate verification failed - Invalid Signature
	KEYM_E_CERTIFICATE_INVALID_CHAIN_OF_TRUST	0x07	Certificate verification failed - Invalid Chain of Trust

	KEYM_E_CERTIFICATE_INVALID_ TYPE	0x08	Certificate verification failed - Invalid Type
	KEYM_E_CERTIFICATE_INVALID_ FORMAT	0x09	Certificate verification failed - Invalid Format
	KEYM_E_CERTIFICATE_INVALID_ CONTENT	0x0A	Certificate verification failed - Invalid Content
	KEYM_E_CERTIFICATE_REVOKED	0x0B	Certificate verification failed - Invalid Scope
Description	Enumeration of the result type of verification operations.		
Variation	--		
Available via	Rte_KeyM_Type.h		

⌋()

8.7.2.7 KeyM_CertificateElementType_{ KeyMCertificate }_{ KeyMCertificateElement }

[SWS_KeyM_91004]⌈

Name	KeyM_CertificateElementType_{KeyMCertificate}_{KeyMCertificateElement}		
Kind	Array	***Element type***	uint8
Size	{ecuc(KeyM/KeyMCertificateElement/KeyMCertificateElementMaxLength} Elements		
Description	Array long enough to store data		

Variation	KeyMCertificate ={ecuc(KeyM/KeyMCertificate.SHORT-NAME)}
	KeyMCertificateElement ={ecuc(KeyM/KeyMCertificate/KeyMCertificate Element.SHORT-NAME)}		

Available via	Rte_KeyM_Type.h		

⌋()

8.7.2.8 KeyM_CryptoKeyDataType

[SWS_KeyM_91012]⌈

Name	KeyM_CryptoKeyDataType
Kind	Pointer
Type	uint8*
Description	Byte-pointer to the input or output data
Variation	--
Available via	Rte_KeyM_Type.h

⌋()

8.7.2.9 KeyM_ResultType

[SWS_KeyM_91008]⌈

Name	KeyM_ResultType		
Kind	Enumeration		
Range	KEYM_RT_OK	0x00	Key management operation successful.
	KEYM_RT_NOT_OK	0x01	General error occured during key management operation.
	KEYM_RT_KEY_CERT_ INVALID	0x02	Key or certificate is invalid and cannot be used for the operation.
	KEYM_RT_KEY_CERT_ WRITE_FAIL	0x03	Key or certificate could not be written to designated storage.
	KEYM_RT_KEY_CERT_ UPDATE_FAIL	0x04	General failure while updating a key or certificate (error code could not be precised by one of the other error codes)
	KEYM_RT_CERT_ INVALID_CHAIN_OF_ TRUST	0x05	Certificate verification failed - Invalid Chain of Trust
Description	Specifies the result type of an asynchronous key management function.		
Variation	--		

Available via	Rte_KeyM_Type.h

⌋()

8.7.2.10 KeyM_CertDataType

[SWS_KeyM_00041]⌈

Name	KeyM_CertDataType	
Kind	Structure	
Elements	certDataLength	
	Type	uint32
	Comment	Length of the certificate data.
	certData	
	Type	VoidPtr
	Comment	Pointer references the data for a certificate on a local data area of the caller.
Description	This structure is used to exchange certificate data through interface functions.	
Variation	--	
Available via	KeyM.h	

⌋()

8.7.3 Client-Server-Interfaces

8.7.3.1 KeyM_Certificate

[SWS_KeyM_00082]⌈

Name	KeyMCertificate_{KeyMCertificate}

Comment	Service of Certificate sub module		
IsService	true		
Variation	KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)}		
Possible Errors	0	E_OK	--
	1	E_NOT_OK	--
	2	KEYM_E_BUSY	--
	4	KEYM_E_KEY_CERT_SIZE_MISMATCH	--
	5	KEYM_E_PARAMETER_MISMATCH	--
	7	KEYM_E_KEY_CERT_WRITE_FAIL	--
	9	KEYM_E_KEY_CERT_READ_FAIL	--
	10	KEYM_E_KEY_CERT_EMPTY	--
	11	KEYM_E_CERT_INVALID_CHAIN_OF_TRUST	--

Operation	GetCertificate	
Comment	Read certificate data from the certificate sub module	
Variation	--	
Parameters	Certificate	
	Type	KeyM_CertDataType
	Direction	OUT
	Comment	Certificate
	Variation	KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)}
Possible Errors	E_OK E_NOT_OK	

	KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH KEYM_E_KEY_CERT_READ_FAIL KEYM_E_KEY_CERT_EMPTY

Operation	GetStatus	
Comment	Provides the status of a certificate.	
Variation	--	
Parameters	Status	
	Type	KeyM_CertificateStatusType
	Direction	OUT
	Comment	Provides the status type.
	Variation	--
Possible Errors	E_OK E_NOT_OK	

Operation	SetCertificate	
Comment	Provides certificate data to be processed by the certificate sub module	
Variation	--	
Parameters	Certificate	
	Type	KeyM_CertDataType
	Direction	IN
	Comment	Certificate data
	Variation	KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)}
Possible Errors	E_OK E_NOT_OK	

	KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH KEYM_E_KEY_CERT_WRITE_FAIL

Operation	VerifyCertificate
Comment	Verify certificate data from the certificate sub module
Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_BUSY KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH KEYM_E_KEY_CERT_EMPTY KEYM_E_CERT_INVALID_CHAIN_OF_TRUST

⌋()

8.7.3.2 KeyMCertificateNotification

[SWS_KeyM_00159]⌈

Name	KeyMCertificateNotification		
Comment	This service interface provides callbacks for certificate management operation.		
IsService	true		
Variation	--		
Possible Errors	--	--	--

Operation	{ecuc(KeyM/KeyMGeneral/KeyMServiceCertificateFunctionEnabled)} == true	
Comment	Notifies the application that a certificate verification has been finished.	
Variation	{ecuc(KeyM/KeyMGeneral/KeyMServiceCertificateFunctionEnabled)} == true	
Parameters	Result	
	Type	KeyM_CertificateStatusType

	Direction	IN
	Comment	Contains information about the result of the operation.
	Variation	--
Possible Errors	--	

Operation	ServiceCertificateCallbackNotification	
Comment	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.	
Variation	--	
Parameters	Result	
	Type	KeyM_ResultType
	Direction	IN
	Comment	Contains information about the result of the operation.
	Variation	--
	ResponseDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	
	Type	KeyM_CryptoKeyDataType

	Direction	IN
	Comment	Data returned by this operation
	Variation	--
Possible Errors	--	

⌋()

8.7.3.3 KeyMCertificateElement

[SWS_KeyM_00083]⌈

Name	KeyMCertificateElement_{KeyMCertificate}_{KeyMCertificateElement}		
Comment	Service of the certificate sub module to access certificate elements.		
IsService	true		
Variation	KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)} KeyMCertificate Element ={ecuc(KeyM/KeyMCertificate/KeyMCertificateElement.SHORT-NAME)}		
Possible Errors	0	E_OK	--
	1	E_NOT_OK	--
	4	KEYM_E_KEY_CERT_SIZE_MISMATCH	--
	5	KEYM_E_PARAMETER_MISMATCH	--
	6	KEYM_E_CERT_INVALID	--
	10	KEYM_E_KEY_CERT_EMPTY	--

Operation	CertificateElementGet
Comment	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.

Variation	--	
Parameters	CertificateElementData	
	Type	KeyM_CertificateElementType_{KeyMCertificate}_{KeyMCertificate-Element}
	Direction	OUT
	Comment	--
	Variation	KeyMCertificate ={ecuc(KeyM/KeyMCertificate.SHORT-NAME)}, Key MCertificateElement ={ecuc(KeyM/KeyMCertificate/KeyMCertificate Element.SHORT-NAME)}
	CertificateDataLength	
	Type	uint32
	Direction	OUT
	Comment	--
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH KEYM_E_CERT_INVALID KEYM_E_KEY_CERT_EMPTY	

Operation	CertificateElementGetByIndex	
Comment	This operation provides the data of a certificate element. The function is used when an element may contain more than one element. The index allows to access the n(th) value of an element. This can be considered like an "array" access. Index=0 accesses the first element.	
Variation	--	
Parameters	Index	
	Type	uint16

	Direction	IN
	Comment	This is the index to dedicated element in the list
	Variation	--
	CertificateElementData	
	Type	KeyM_CertificateElementType_{KeyMCertificate}_{KeyMCertificate-Element}
	Direction	OUT
	Comment	--
	Variation	KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)}, Key MCertificateElement ={ecuc(KeyM/KeyMCertificate/KeyMCertificate Element.SHORT-NAME)}
	CertificateDataLength	
	Type	uint32
	Direction	OUT
	Comment	--
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH KEYM_E_CERT_INVALID KEYM_E_KEY_CERT_EMPTY	

Operation	CertificateElementGetCount
Comment	This operation provides the amount of data elements available for the certificate element. This function is useful to retrieve the total amount of data elements available in one certificate element and is used in combination with the operation Certificate ElementGetByIndex. If only one data element is available, the function returns "1".
Variation	--

Parameters	count	
	Type	uint16
	Direction	OUT
	Comment	Number of items available for an element
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH KEYM_E_CERT_INVALID KEYM_E_KEY_CERT_EMPTY	

⌋ ( )

8.7.3.4 KeyMCryptoKey

[SWS_KeyM_00084]⌈

Name	KeyMCryptoKey		
Comment	Service of CryptoKey sub module		
IsService	true		
Variation	--		
Possible Errors	0	E_OK	--
	1	E_NOT_OK	--
	2	KEYM_E_BUSY	--
	3	KEYM_E_PENDING	--
	4	KEYM_E_KEY_CERT_SIZE_MISMATCH	--
	5	KEYM_E_PARAMETER_MISMATCH	--
	6	KEYM_E_CERT_INVALID	--

	10	KEYM_E_KEY_CERT_EMPTY	--

Operation	Finalize	
Comment	--	
Variation	{ecuc(KeyM/KeyMGeneral/KeyMCryptoKeyHandlerStartFinalizeEnabled)} == true	
Parameters	RequestData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Information that comes along with the request, e.g. signature
	Variation	--
	RequestDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	
	Type	KeyM_CryptoKeyDataType
	Direction	OUT
	Comment	Data returned by this operation
	Variation	--
	ResponseDataLength	
	Type	uint16

	Direction	OUT
	Comment	--
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH	

Operation	Prepare	
Comment	--	
Variation	{ecuc(KeyM/KeyMGeneral/KeyMCryptoKeyPrepareFunctionEnabled)} == true	
Parameters	RequestData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Information that comes along with the request, e.g. signature
	Variation	--
	RequestDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	
	Type	KeyM_CryptoKeyDataType
	Direction	OUT

	Comment	Data returned by this operation
	Variation	--
	ResponseDataLength	
	Type	uint16
	Direction	OUT
	Comment	--
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH	

Operation	Start	
Comment	This function intents to start a key update operation.	
Variation	{ecuc(KeyM/KeyMGeneral/KeyMCryptoKeyHandlerStartFinalizeEnabled)} == true	
Parameters	StartType	
	Type	KeyM_StartType
	Direction	IN
	Comment	Defines in which mode the key operation shall be executed
	Variation	--
	RequestData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Information that comes along with the request, e.g. signature

	Variation	--
	RequestDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	
	Type	KeyM_CryptoKeyDataType
	Direction	OUT
	Comment	Data returned by this operation
	Variation	--
	ResponseDataLength	
	Type	uint16
	Direction	OUT
	Comment	--
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH	

Operation	Update
Comment	--
Variation	--

Parameters	KeyName	
	Type	KeyM_KeyCertNameDataType
	Direction	IN
	Comment	Provides the name of the key that shall be verified
	Variation	--
	KeyNameLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	RequestData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Information that comes along with the request, e.g. signature
	Variation	--
	RequestDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	

	Type	KeyM_CryptoKeyDataType
	Direction	OUT
	Comment	Data returned by this operation
	Variation	--
	ResponseDataLength	
	Type	uint16
	Direction	OUT
	Comment	--
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH	

Operation	Verify	
Comment	The intention is to perform a verification of input data using an assigned crypto job with its key.	
Variation	{ecuc(KeyM/KeyMGeneral/KeyMCryptoKeyVerifyFunctionEnabled)} == true	
Parameters	KeyName	
	Type	KeyM_KeyCertNameDataType
	Direction	IN
	Comment	Provides the name of the key that shall be verified
	Variation	--
	KeyNameLength	

	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	RequestData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Information that comes along with the request, e.g. signature
	Variation	--
	RequestDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	
	Type	KeyM_CryptoKeyDataType
	Direction	OUT
	Comment	Data returned by this operation
	Variation	--
	ResponseDataLength	
	Type	uint16

	Direction	OUT
	Comment	--
	Variation	--
Possible Errors	E_OK E_NOT_OK KEYM_E_BUSY KEYM_E_PENDING KEYM_E_KEY_CERT_SIZE_MISMATCH KEYM_E_PARAMETER_MISMATCH KEYM_E_CERT_INVALID KEYM_E_KEY_CERT_EMPTY	

⌋()

8.7.3.5 KeyMCryptoKeyNotification

[SWS_KeyM_91005]⌈

Name	KeyMCryptoKeyNotification		
Comment	Service of <module>		
IsService	true		
Variation	--		
Possible Errors	--	--	--

Operation	CryptoKeyFinalizeCallbackNotification	
Comment	Notifies the application that a crypto key finalize operation has been finished. The callback function is only called and needed if KeyMCryptoKeyStartFinalize FunctionEnabled is set to TRUE.	
Variation	{ecuc(KeyM/KeyMGeneral/KeyMCryptoKeyStartFinalizeFunctionEnabled)} == true	
Parameters	Result	
	Type	KeyM_ResultType
	Direction	IN

	Comment	Contains information about the result of the operation.
	Variation	--
	ResponseDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Data returned by this operation
	Variation	--
Possible Errors	--	

Operation	CryptoKeyUpdateCallbackNotification	
Comment	Notifies the application that a crypto key update operation has been finished. This function is used by the key manager.	
Variation	--	
Parameters	Result	
	Type	KeyM_ResultType
	Direction	IN
	Comment	Contains information about the result of the operation.

	Variation	--
	ResponseDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResponseData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Data returned by this operation
	Variation	--
Possible Errors	--	

Operation	CryptoKeyVerifyCallbackNotification	
Comment	Notifies the application that a crypto key verify operation has been finished. This function is used by the key manager.	
Variation	--	
Parameters	Result	
	Type	KeyM_ResultType
	Direction	IN
	Comment	Contains information about the result of the operation.
	Variation	--

	KeyId	
	Type	uint32
	Direction	IN
	Comment	The key identifier where this verification was started for.
	Variation	--
	ResultDataLength	
	Type	uint16
	Direction	IN
	Comment	--
	Variation	--
	ResultData	
	Type	KeyM_CryptoKeyDataType
	Direction	IN
	Comment	Data returned by this operation
	Variation	--
Possible Errors	--	

⌋()

8.7.4 Ports

8.7.4.1 KeyM_Certificate_{KeyMCertificate}

[SWS_KeyM_00160]⌈

Name	KeyMCertificate_{KeyMCertificate}		
Kind	ProvidedPort	***Interface***	KeyMCertificate_{KeyMCertificate}
Description	Port to execute certificate related functions.		
Port Defined Argument Value(s)	***Type***	KeyM_CertificateIdType	
	Value	{ecuc(KeyM/KeyMCertificate/KeyMCertificateId)}	
Variation	KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)}		

⌋(SRS_CryptoStack_00090, SRS_CryptoStack_00091)

8.7.4.2 KeyM_CertificateNotification_{KeyMCertificate}

[SWS_KeyM_00161]⌈

Name	KeyMCertificateNotification_{KeyMCertificate}		
Kind	RequiredPort	***Interface***	KeyMCertificateNotification
Description	Port to execute certificate notification related functions.		
Port Defined Argument Value(s)	***Type***	KeyM_CertificateIdType	
	Value	{ecuc(KeyM/KeyMCertificate/KeyMCertificateId)}	
Variation	KeyMCertificateVerifyCallbackNotificationFunc == NULL KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)}		

⌋(SRS_CryptoStack_00090, SRS_CryptoStack_00091)

8.7.4.3 KeyMCertificateElement_{KeyMCertificate}_{KeyMCertificateElement}

[SWS_KeyM_00162]⌈

Name	KeyMCertificateElement_{KeyMCertificate}_{KeyMCertificateElement}		
Kind	ProvidedPort	***Interface***	KeyMCertificateElement_{KeyMCertificate}_{-KeyMCertificateElement}
Description	Port to execute certificate related functions.		

Port Defined Argument Value(s)	***Type***	KeyM_CertificateIdType
	Value	{ecuc(KeyM/KeyMCertificate/KeyMCertificateId)}
	Type	KeyM_CertElementIdType
	Value	{ecuc(KeyM/KeyMCertificate/KeyMCertificateElement/Key MCertificateElementId)}
Variation	KeyMCertificate = {ecuc(KeyM/KeyMCertificate.SHORT-NAME)} Key MCertificateElement = {ecuc(KeyM/KeyMCertificate/KeyMCertificate Element.SHORT-NAME)}	

⌋(SRS_CryptoStack_00090, SRS_CryptoStack_00091)

8.7.4.4 KeyMCryptoKey

[SWS_KeyM_00163]⌈

Name	KeyMCryptoKey		
Kind	ProvidedPort	***Interface***	KeyMCryptoKey
Description	Port to execute crypto key related functions.		
Variation	--		

⌋(SRS_CryptoStack_00090, SRS_CryptoStack_00091)

8.7.4.5 KeyMCryptoKeyNotification

[SWS_KeyM_00164]⌈

Name	KeyMCryptoKeyNotification		
Kind	RequiredPort	***Interface***	KeyMCryptoKeyNotification
Description	Port to execute crypto key notification related functions.		
Variation	--		

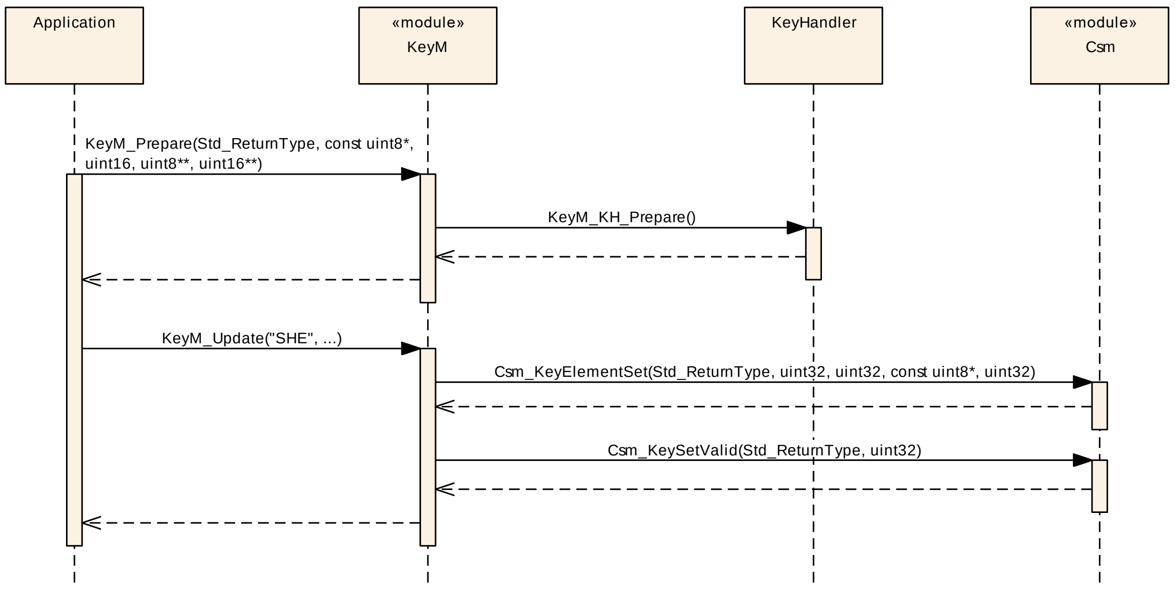
⌋(SRS_CryptoStack_00090, SRS_CryptoStack_00091)

9 Sequence diagrams

9.1 Store single key

Configuration item *KeyMCryptoKeyStartFinalizeFunctionEnabled* assumed to be `FALSE`, `KeyM_Prepare()` is activated and delegated to the key handler. `KeyM_Update()` operation completely covered by KeyM.

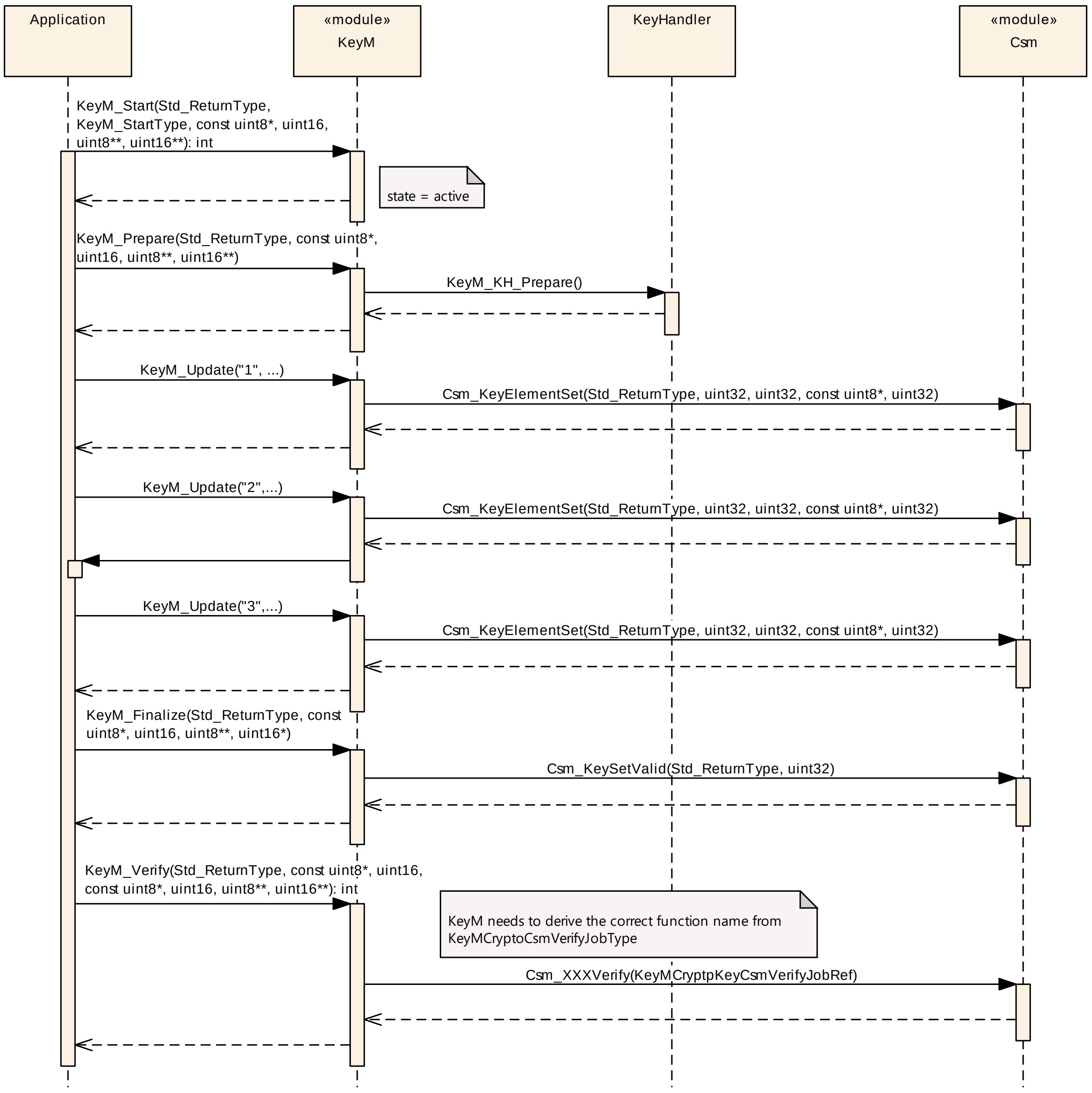
Store single key sequence (KeyMCryptoKeyGenerationType==KEYM_STORED_KEY)


Application
«module» KeyM
KeyHandler
«module» Csm
KeyM_Prepare(Std_ReturnType, const uint8*, uint16, uint8**, uint16**)
KeyM_KH_Prepare()
KeyM_Update("SHE", ...)
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
Csm_KeySetValid(Std_ReturnType, uint32)

9.2 Store multiple keys

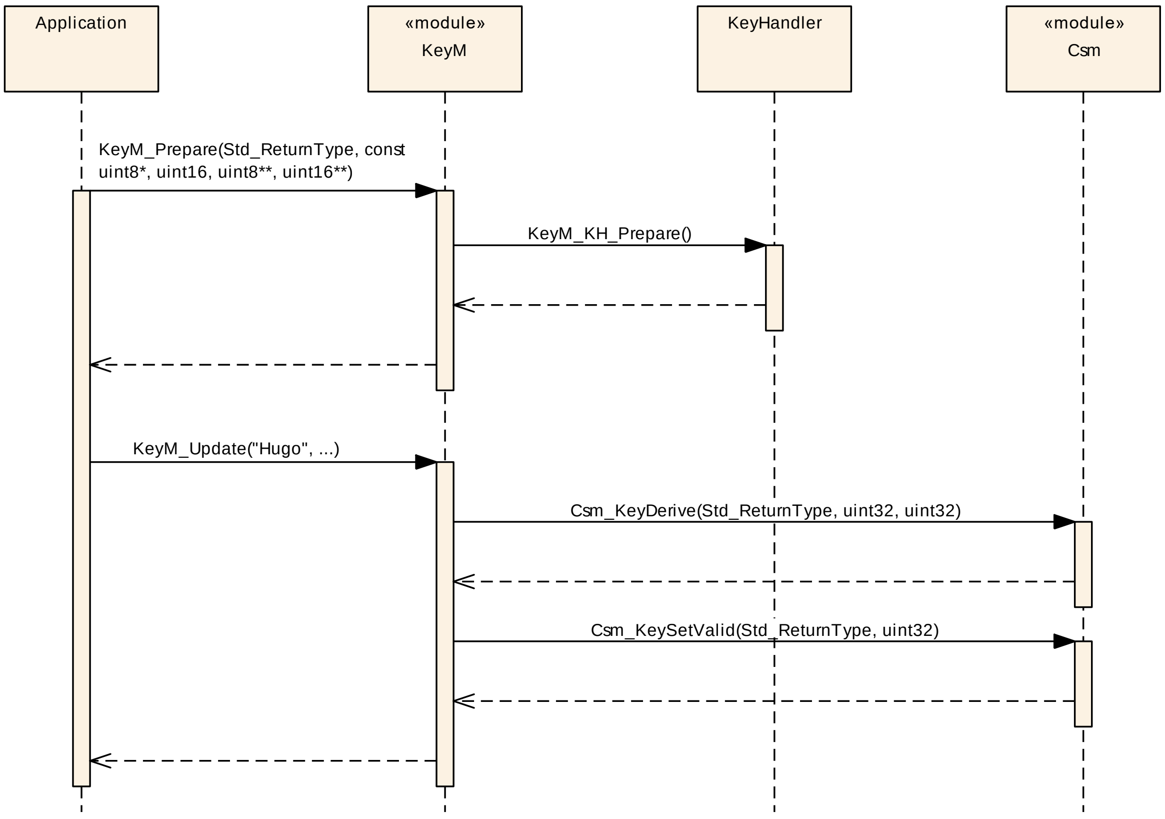
Example with StartFinalize enabled and managed by KeyM (no delegation via `KeyM_KH_Start()` to key handler). The `KeyM_Prepare()` operation is delegated to the key handler. Multiple keys are set or updated using multiple `KeyM_Update()` calls. The keys are updated using the `Csm_KeyElementSet()` function according to the configuration of the keys.
During finalization the KeyM sets all keys to valid.

Store multiple keys sequence (KeyMCryptoKeyGenerationType==KEYM_STORED_KEY)

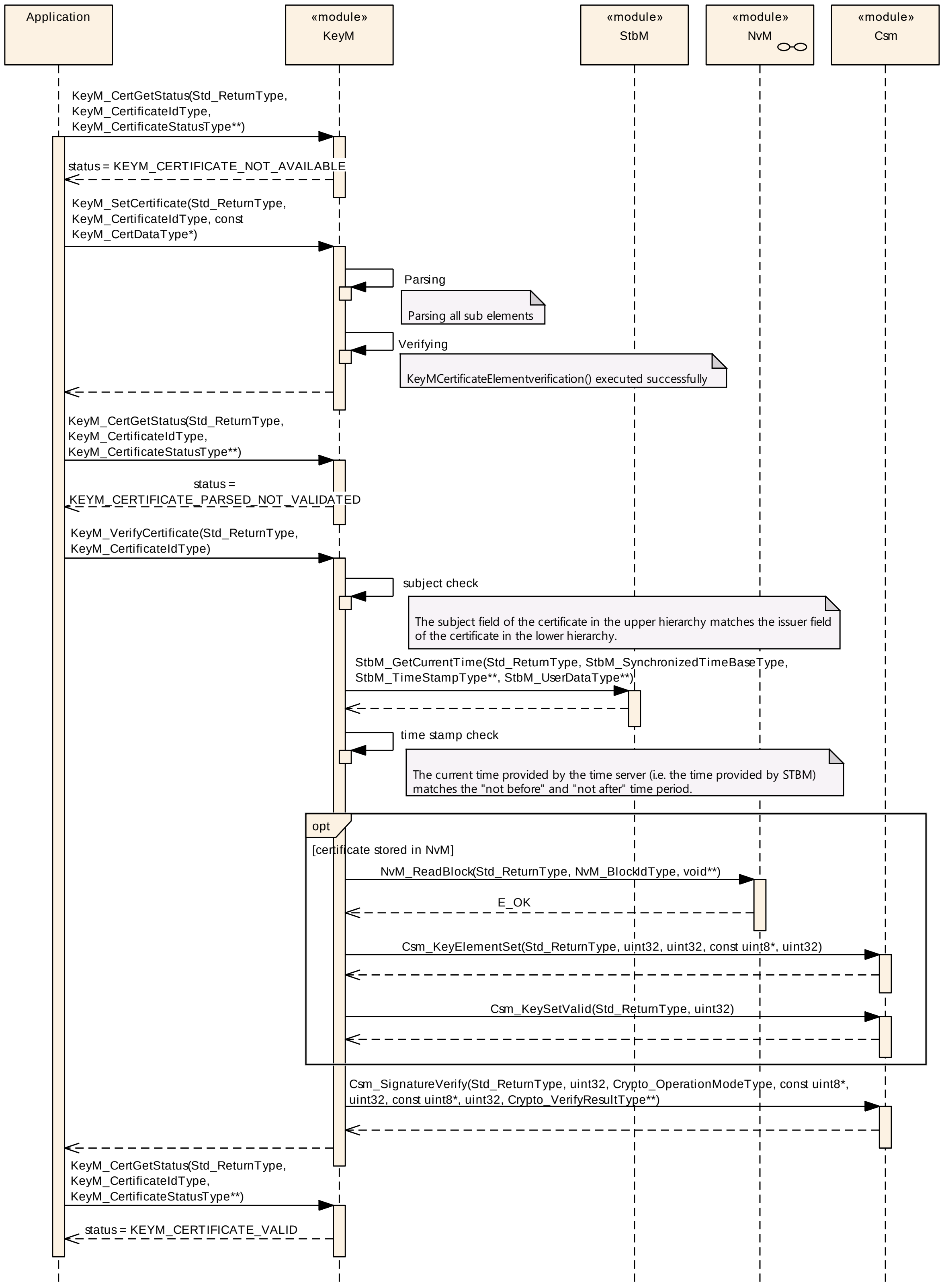

Application
«module» KeyM
KeyHandler
«module» Csm
KeyM_Start(Std_ReturnType, KeyM_StartType, const uint8*, uint16, uint8**, uint16**): int
state = active
KeyM_Prepare(Std_ReturnType, const uint8*, uint16, uint8**, uint16**)
KeyM_KH_Prepare()
KeyM_Update("1", ...)
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
KeyM_Update("2",...)
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
KeyM_Update("3",...)
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
KeyM_Finalize(Std_ReturnType, const uint8*, uint16, uint8**, uint16*)
Csm_KeySetValid(Std_ReturnType, uint32)
KeyM_Verify(Std_ReturnType, const uint8*, uint16, const uint8*, uint16, uint8**, uint16**): int
KeyM needs to derive the correct function name from KeyMCryptoCsmVerifyJobType
Csm_XXXVerify(KeyMCryptpKeyCsmVerifyJobRef)

9.3 Derive key

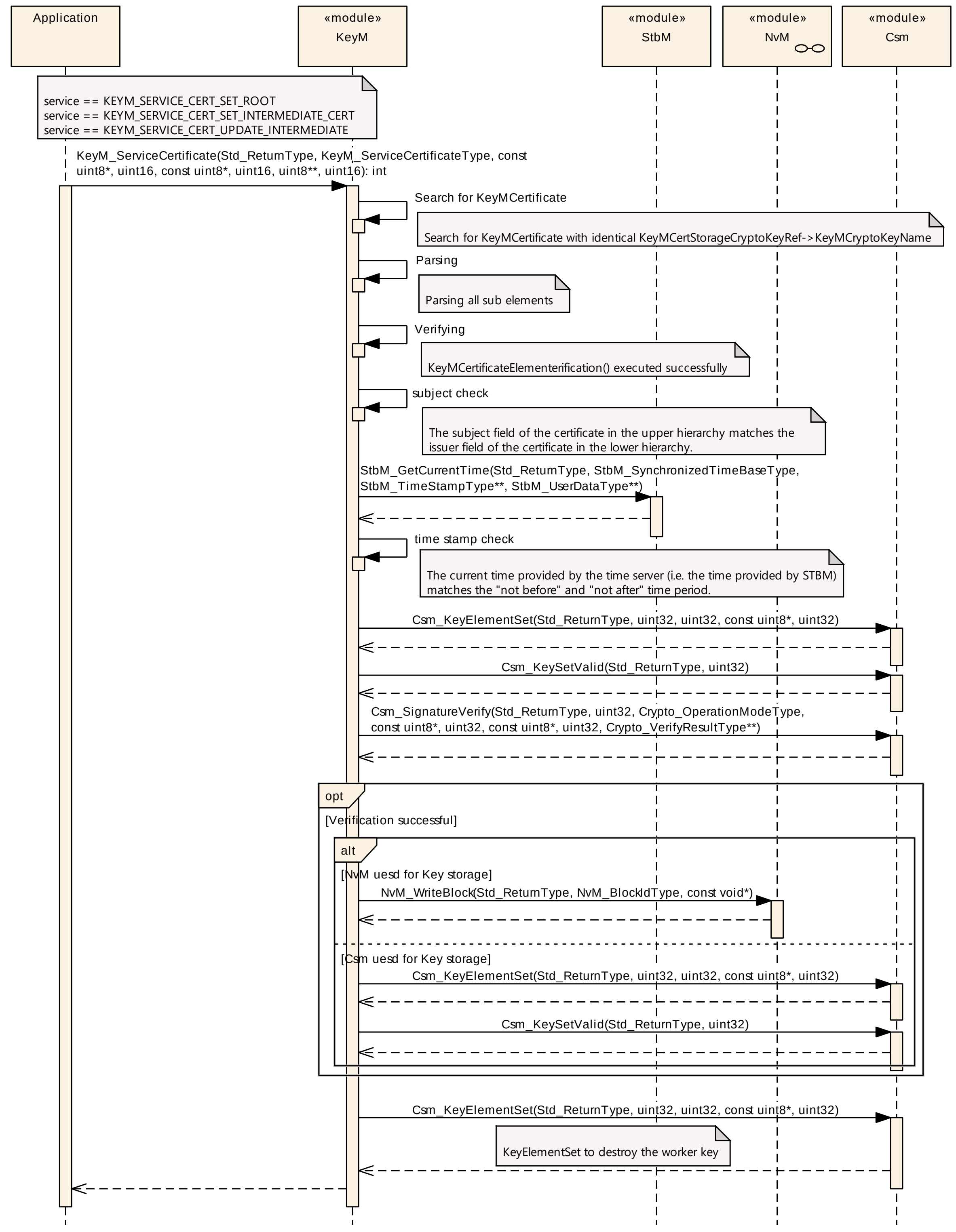
Example using Csm_KeyDerive sequence instead of Csm_KeyElementSet() **(KeyMCryptoKeyGenerationType==KEYM_DERIVED_KEY)**.


Application
«module»
KeyM
KeyHandler
«module»
Csm
KeyM_Prepare(Std_ReturnType, const uint8*, uint16, uint8**, uint16**)
KeyM_KH_Prepare()
KeyM_Update("Hugo", ...)
Csm_KeyDerive(Std_ReturnType, uint32, uint32)
Csm_KeySetValid(Std_ReturnType, uint32)

9.4 Add working certificate


Application
«module» KeyM
«module» StbM
«module» NvM
«module» Csm
KeyM_CertGetStatus(Std_ReturnType, KeyM_CertificateIdType, KeyM_CertificateStatusType**)
status = KEYM_CERTIFICATE_NOT_AVAILABLE
KeyM_SetCertificate(Std_ReturnType, KeyM_CertificateIdType, const KeyM_CertDataType*)
Parsing
Parsing all sub elements
Verifying
KeyMCertificateElementverification() executed successfully
KeyM_CertGetStatus(Std_ReturnType, KeyM_CertificateIdType, KeyM_CertificateStatusType**)
status = KEYM_CERTIFICATE_PARSED_NOT_VALIDATED
KeyM_VerifyCertificate(Std_ReturnType, KeyM_CertificateIdType)
subject check
The subject field of the certificate in the upper hierarchy matches the issuer field of the certificate in the lower hierarchy.
StbM_GetCurrentTime(Std_ReturnType, StbM_SynchronizedTimeBaseType, StbM_TimeStampType**, StbM_UserDataType**)
time stamp check
The current time provided by the time server (i.e. the time provided by STBM) matches the "not before" and "not after" time period.
opt
[certificate stored in NvM]
NvM_ReadBlock(Std_ReturnType, NvM_BlockIdType, void**)
E_OK
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
Csm_KeySetValid(Std_ReturnType, uint32)
Csm_SignatureVerify(Std_ReturnType, uint32, Crypto_OperationModeType, const uint8*, uint32, const uint8*, uint32, Crypto_VerifyResultType**)
KeyM_CertGetStatus(Std_ReturnType, KeyM_CertificateIdType, KeyM_CertificateStatusType**)
status = KEYM_CERTIFICATE_VALID

9.5 Add root or intermediate certificate


Application
«module» KeyM
«module» StbM
«module» NvM
«module» Csm
service == KEYM_SERVICE_CERT_SET_ROOT
service == KEYM_SERVICE_CERT_SET_INTERMEDIATE_CERT
service == KEYM_SERVICE_CERT_UPDATE_INTERMEDIATE
KeyM_ServiceCertificate(Std_ReturnType, KeyM_ServiceCertificateType, const uint8*, uint16, const uint8*, uint16, uint8**, uint16): int
Search for KeyMCertificate
Search for KeyMCertificate with identical KeyMCertStorageCryptoKeyRef->KeyMCryptoKeyName
Parsing
Parsing all sub elements
Verifying
KeyMCertificateElementerification() executed successfully
subject check
The subject field of the certificate in the upper hierarchy matches the issuer field of the certificate in the lower hierarchy.
StbM_GetCurrentTime(Std_ReturnType, StbM_SynchronizedTimeBaseType, StbM_TimeStampType**, StbM_UserDataType**)
time stamp check
The current time provided by the time server (i.e. the time provided by STBM) matches the "not before" and "not after" time period.
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
Csm_KeySetValid(Std_ReturnType, uint32)
Csm_SignatureVerify(Std_ReturnType, uint32, Crypto_OperationModeType, const uint8*, uint32, const uint8*, uint32, Crypto_VerifyResultType**)
opt
[Verification successful]
alt
[NvM uesd for Key storage]
NvM_WriteBlock(Std_ReturnType, NvM_BlockIdType, const void*)
[Csm uesd for Key storage]
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
Csm_KeySetValid(Std_ReturnType, uint32)
Csm_KeyElementSet(Std_ReturnType, uint32, uint32, const uint8*, uint32)
KeyElementSet to destroy the worker key

10 Configuration specification

Chapter 10.1 specifies the structure (containers) and the parameters of the module KeyM.

Chapter 10.2 specifies additionally published information of the module KeyM.

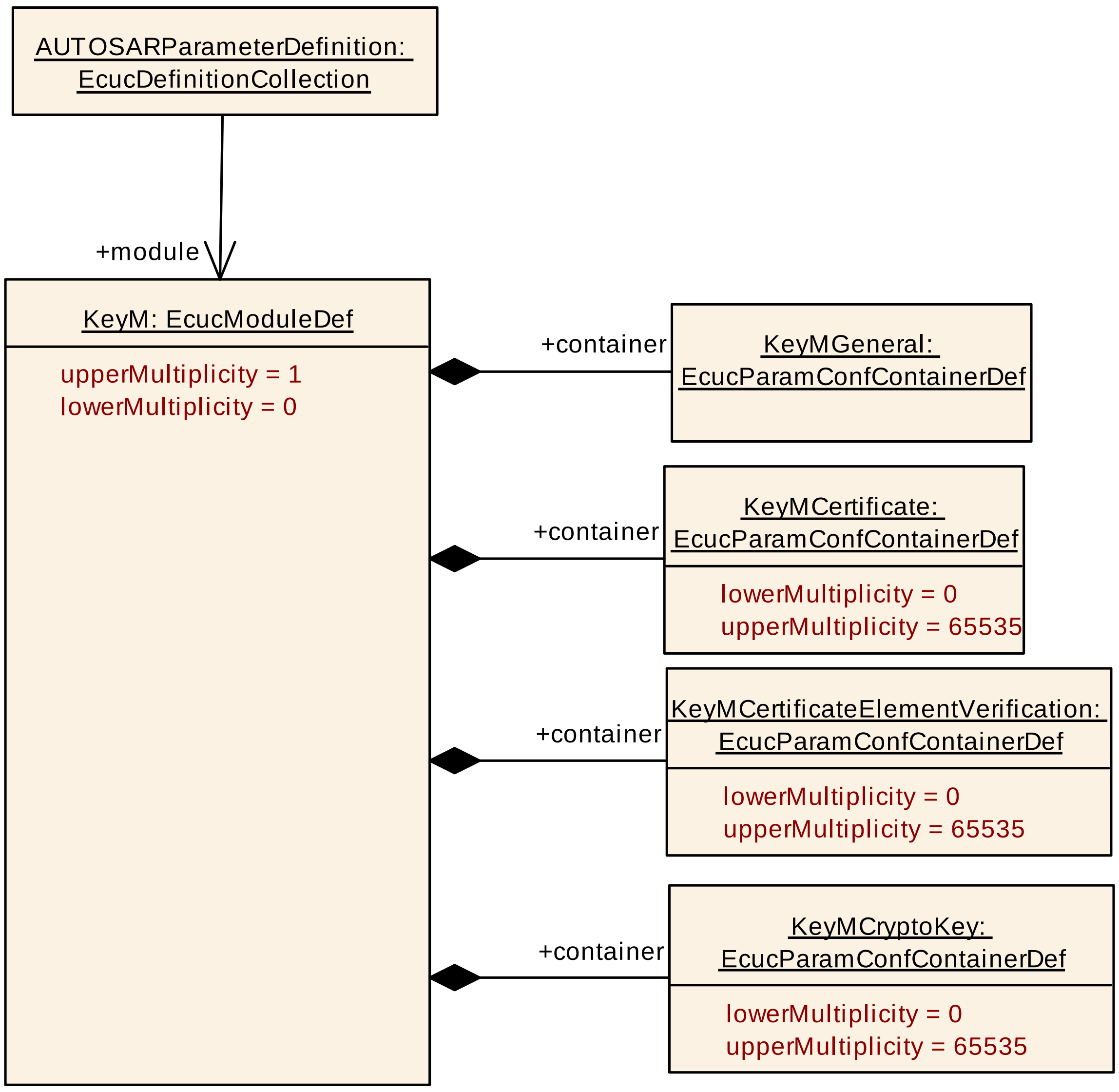
10.1 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

10.1.1 KeyM

SWS Item	**ECUC_KeyM_00001 :**
Module Name	*KeyM*
Module Description	Configuration of the Mcu (Microcontroller Unit) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
KeyMCertificate	0..65535	This container contains the certificate configuration.
KeyMCertificateElementVerification	0..65535	This container defines if and how certificate elements are to be verified.
KeyMCryptoKey	0..65535	This container contains the crypto keys that can be updated.
KeyMGeneral	1	This container holds general configuration (parameters) for key manager.
KeyMNvmBlock	0..65535	Configuration of optional usage of Nvm in case the KeyM module requires non volatile memory in the Ecu to store information (e.g. crypto keys or certificates).


AUTOSARParameterDefinition: EcucDefinitionCollection
+module
KeyM: EcucModuleDef
upperMultiplicity = 1
lowerMultiplicity = 0
+container
KeyMGeneral: EcucParamConfContainerDef
+container
KeyMCertificate: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+container
KeyMCertificateElementVerification: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+container
KeyMCryptoKey: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535

10.1.2 KeyMGeneral

SWS Item	**ECUC_KeyM_00002 :**
Container Name	KeyMGeneral
Parent Container	KeyM
Description	This container holds general configuration (parameters) for key manager.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00008 :**		
Name	KeyMCertificateChainMaxDepth		
Parent Container	KeyMGeneral		
Description	Maximum number of certificates defined in a certificate chain.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00010 :**		
Name	KeyMCertificateManagerEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the part that manages certificates.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00018 :**		
Name	KeyMCryptoKeyHandlerPrepareEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the key handler prepare function call. If set to true, the corresponding key handler function shall be provided.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00021 :**		
Name	KeyMCryptoKeyHandlerServiceCertificateEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the key handler service function call. If set to true, the certificate submodule function KeyM_KH_ServiceCertificate() shall be provided.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00017 :**
Name	KeyMCryptoKeyHandlerStartFinalizeEnabled
Parent Container	KeyMGeneral
Description	Enables (TRUE) or disables (FALSE) the key handler start and finalize function call. If set to true, the key handler functions KeyM_KH_Start() and KeyM_KH_Finalize() shall be provided.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Post-Build Variant Value	false

Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00019 :**		
Name	KeyMCryptoKeyHandlerUpdateEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the call to the key handler update function KeyM_KH_Update(). If set to true, the corresponding key handler function shall be provided.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00020 :**		
Name	KeyMCryptoKeyHandlerVerifyEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the call to the key handler verify function KeyM_KH_Verify(). If set to true, the corresponding key handler function shall be provided.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00011 :**		
Name	KeyMCryptoKeyManagerEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the part that manages crypto key operations.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00013 :**
Name	KeyMCryptoKeyPrepareFunctionEnabled
Parent Container	KeyMGeneral
Description	Enables (TRUE) or disables (FALSE) the prepare function of the key manager. If set to true, the KeyM_Prepare() function has to be called accordingly.
Multiplicity	0..1

Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00012 :**		
Name	KeyMCryptoKeyStartFinalizeFunctionEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the start and Finish function of the key manager. If set to true, the KeyM_Start() and KeyM_Finalize() functions have to be called.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00015 :**		
Name	KeyMCryptoKeyVerifyAsyncMode		
Parent Container	KeyMGeneral		
Description	This parameter defines if the function KeyM_Verify() runs in synchronous or asynchronous mode		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00014 :**
Name	KeyMCryptoKeyVerifyFunctionEnabled
Parent Container	KeyMGeneral

Description	Enables (TRUE) or disables (FALSE) the verify function of the key manager. If set to true, the KeyM_Verify() function can be called.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00006 :**		
Name	KeyMDevErrorDetect		
Parent Container	KeyMGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00078 :**		
Name	KeyMEnableSecurityEventReporting		
Parent Container	KeyMGeneral		
Description	Switches the reporting of security events to the IdsM: - true: reporting is enabled. - false: reporting is disabled. **Tags:** atp.Status=draft		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

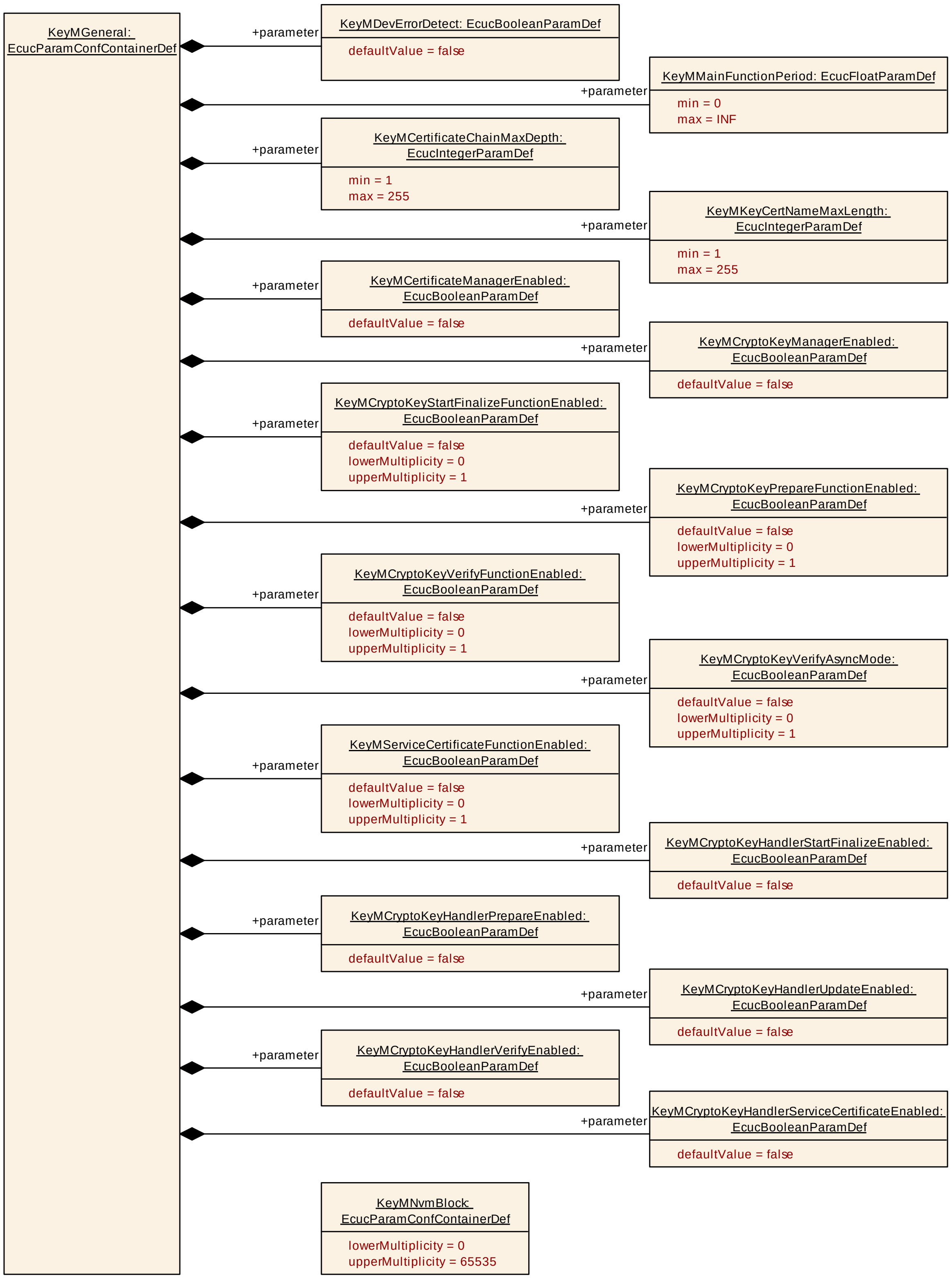
SWS Item	**ECUC_KeyM_00009 :**
Name	KeyMKeyCertNameMaxLength
Parent Container	KeyMGeneral
Description	Maximum length in bytes of certificate or key names used for the service interface.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00007 :**		
Name	KeyMMainFunctionPeriod		
Parent Container	KeyMGeneral		
Description	Specifies the period of main function KeyM_MainFunction in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00016 :**		
Name	KeyMServiceCertificateFunctionEnabled		
Parent Container	KeyMGeneral		
Description	Enables (TRUE) or disables (FALSE) the certificate service function of the key manager. If set to true, the KeyM_ServiceCertificate() function has to be called accordingly.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
KeyMSecurityEventRefs	0..1	Container for the references to IdsMEvent elements representing the security events that the KeyM module shall report to the IdsM in case the coresponding security related event occurs (and if KeyMEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events. **Tags:** atp.Status=draft


KeyMGeneral: EcucParamConfContainerDef
+parameter
KeyMDevErrorDetect: EcucBooleanParamDef
defaultValue = false
KeyMMainFunctionPeriod: EcucFloatParamDef
min = 0
max = INF
KeyMCertificateChainMaxDepth: EcucIntegerParamDef
min = 1
max = 255
KeyMKeyCertNameMaxLength: EcucIntegerParamDef
min = 1
max = 255
KeyMCertificateManagerEnabled: EcucBooleanParamDef
defaultValue = false
KeyMCryptoKeyManagerEnabled: EcucBooleanParamDef
defaultValue = false
KeyMCryptoKeyStartFinalizeFunctionEnabled: EcucBooleanParamDef
defaultValue = false
lowerMultiplicity = 0
upperMultiplicity = 1
KeyMCryptoKeyPrepareFunctionEnabled: EcucBooleanParamDef
defaultValue = false
lowerMultiplicity = 0
upperMultiplicity = 1
KeyMCryptoKeyVerifyFunctionEnabled: EcucBooleanParamDef
defaultValue = false
lowerMultiplicity = 0
upperMultiplicity = 1
KeyMCryptoKeyVerifyAsyncMode: EcucBooleanParamDef
defaultValue = false
lowerMultiplicity = 0
upperMultiplicity = 1
KeyMServiceCertificateFunctionEnabled: EcucBooleanParamDef
defaultValue = false
lowerMultiplicity = 0
upperMultiplicity = 1
KeyMCryptoKeyHandlerStartFinalizeEnabled: EcucBooleanParamDef
defaultValue = false
KeyMCryptoKeyHandlerPrepareEnabled: EcucBooleanParamDef
defaultValue = false
KeyMCryptoKeyHandlerUpdateEnabled: EcucBooleanParamDef
defaultValue = false
KeyMCryptoKeyHandlerVerifyEnabled: EcucBooleanParamDef
defaultValue = false
KeyMCryptoKeyHandlerServiceCertificateEnabled: EcucBooleanParamDef
defaultValue = false
KeyMNvmBlock: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535

10.1.3 KeyMCertificate

SWS Item	**ECUC_KeyM_00003 :**
Container Name	KeyMCertificate

Parent Container	KeyM
Description	This container contains the certificate configuration.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00029 :**		
Name	KeyMCertAlgorithmType		
Parent Container	KeyMCertificate		
Description	Specify in which format the certificate will be provided.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	ECC	--	
	RSA	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00028 :**		
Name	KeyMCertFormatType		
Parent Container	KeyMCertificate		
Description	Specify in which format the certificate will be provided.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CRL	--	
	CVC	--	
	X509	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00022 :**		
Name	KeyMCertificateId		
Parent Container	KeyMCertificate		
Description	Identifier of the certificate. The set of configured identifiers shall be consecutive and gapless.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00023 :**
Name	KeyMCertificateMaxLength
Parent Container	KeyMCertificate
Description	Specify the maximum length in bytes of the certificate.

Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00024 :**		
Name	KeyMCertificateName		
Parent Container	KeyMCertificate		
Description	Provides a unique name of the certificate for identification. The certificate provisional will reference certificates by this unique name.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00073 :**		
Name	KeyMCertificateStorage		
Parent Container	KeyMCertificate		
Description	Specify the storage location of the certificate.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	KEYM_STORAGE_IN_CSM	--	
	KEYM_STORAGE_IN_NVM	--	
	KEYM_STORAGE_IN_RAM	--	
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00025 :**
Name	KeyMCertificateVerifyCallbackNotificationFunc
Parent Container	KeyMCertificate
Description	This parameter provides the function name for the callback <KeyM_CertificateVerifyCallbackNotification>. It indicates if a certificate verification operation was finished and provides its status. If this parameter is omitted, no callback will be provided.
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	--
maxLength	--

minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00026 :**		
Name	KeyMServiceCertificateCallbackNotificationFunc		
Parent Container	KeyMCertificate		
Description	This parameter provides the function name for the service certificate callback <KeyM_ServiceCertificateCallbackNotification>. It indicates if a certificate service operation was finished and provides its status. If this parameter is not set, no callback will be provided.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00034 :**		
Name	KeyMCertCertificateElementRuleRef		
Parent Container	KeyMCertificate		
Description	Reference to certificate element rules which should be verified within the certification validation step.		
Multiplicity	0..65535		
Type	Reference to [KeyMCertificateElementRule]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: Key will be located in RAM if this configuration item is not present.		

SWS Item	**ECUC_KeyM_00077 :**		
Name	KeyMCertCsmSignatureGenerateJobRef		
Parent Container	KeyMCertificate		
Description	Reference to a CSM job to calculate a signature		
Multiplicity	0..1		
Type	Symbolic name reference to [CsmJob]		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This item is only needed if a signature need to be generated for a certificate, e.g. for a certificate signing request (CSR).		

SWS Item	**ECUC_KeyM_00030 :**		
Name	KeyMCertCsmSignatureVerifyJobRef		
Parent Container	KeyMCertificate		
Description	Reference to the CSM job that is used to verify the signature		
Multiplicity	1		
Type	Symbolic name reference to [CsmJob]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00031 :**		
Name	KeyMCertCsmSignatureVerifyKeyRef		
Parent Container	KeyMCertificate		
Description	Reference to the CSM key associated to the CSM signature verify job. The Public Key of this certificate shall be set to the key element (CRYPTO_KE_SIGNATURE_KEY) where the key references to.		
Multiplicity	1		
Type	Symbolic name reference to [CsmKey]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00074 :**		
Name	KeyMCertificateCsmKeyTargetRef		
Parent Container	KeyMCertificate		
Description	Defines a reference to the associated CSM key where the certificate shall be stored to.		
Multiplicity	0..1		
Type	Symbolic name reference to [CsmKey]		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: Only necessary if KeyMCertificateStorage is set to KEYM_STORAGE_IN_CSM		

SWS Item	**ECUC_KeyM_00075 :**		
Name	KeyMCertificateNvmBlockRef		
Parent Container	KeyMCertificate		
Description	Defines a reference to the NvMblock where the certificate is going to be stored.		
Multiplicity	0..1		
Type	Reference to [KeyMNvmBlock]		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	dependency: Only necessary if KeyMCertificateStorage is set to KEYM_STORAGE_IN_NVM		

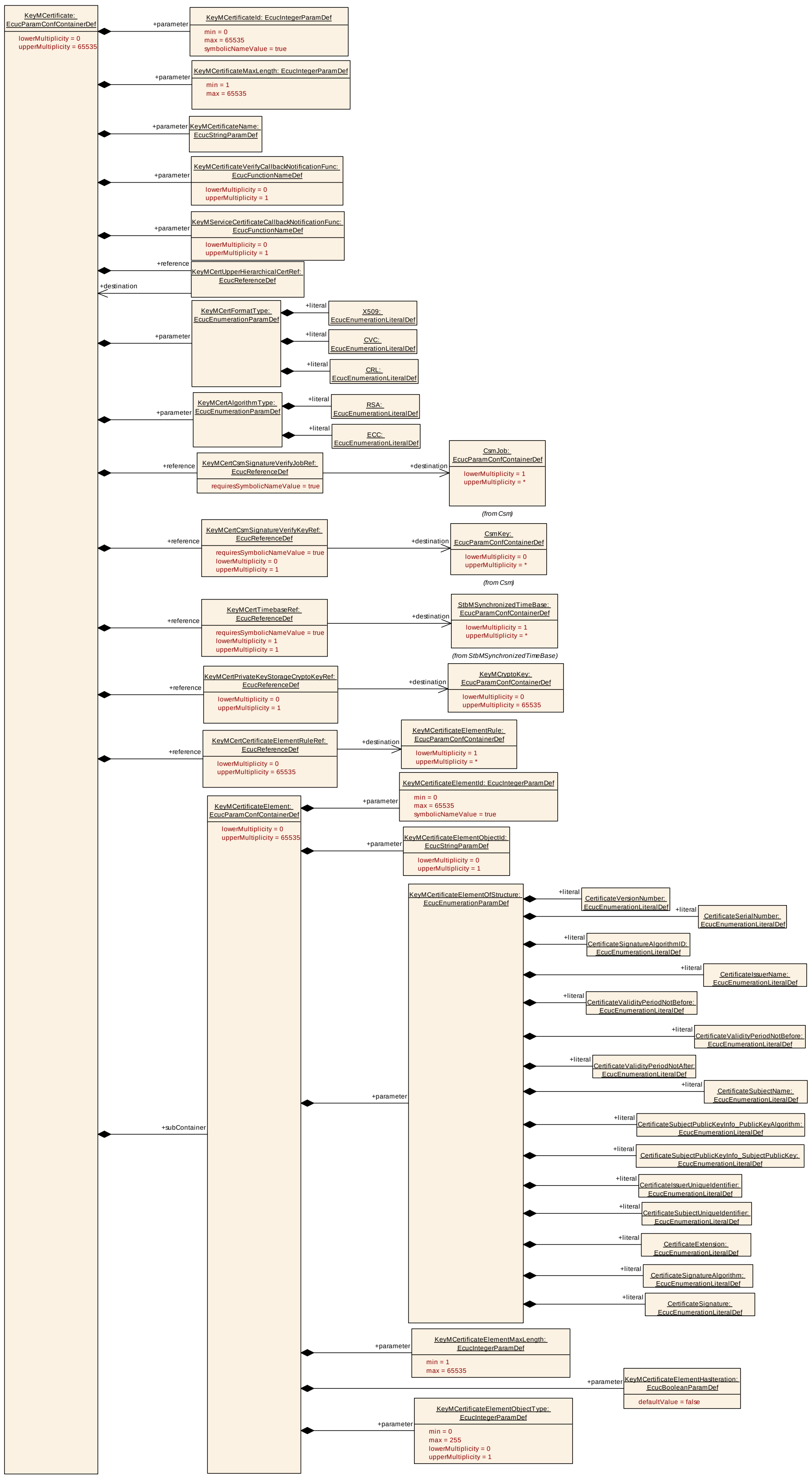
SWS Item	**ECUC_KeyM_00033 :**		
Name	KeyMCertPrivateKeyStorageCryptoKeyRef		
Parent Container	KeyMCertificate		
Description	Defines a storage location of the private key of a certificate.		
Multiplicity	0..1		
Type	Reference to [KeyMCryptoKey]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: Key will be located in RAM if this configuration item is not present.		

SWS Item	**ECUC_KeyM_00032 :**		
Name	KeyMCertTimebaseRef		
Parent Container	KeyMCertificate		
Description	This is a reference to an StbM time base to validate the validity period. Alternatively, KeyMCertificateElementVerification with the KeyMCertificateElement of CertificateValidityPeriodNotBefore or CertificateValidityPeriodNotAfter could be used.		
Multiplicity	1		
Type	Symbolic name reference to [StbMSynchronizedTimeBase]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local dependency: Key will be located in RAM if this configuration item is not present.

SWS Item	**ECUC_KeyM_00027 :**		
Name	KeyMCertUpperHierarchicalCertRef		
Parent Container	KeyMCertificate		
Description	Identifier of the certificate that is the next higher in the PKI hierarchical structure. The reference points to itself for root certificates.		
Multiplicity	1		
Type	Reference to [KeyMCertificate]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
KeyMCertificateElement	0..65535	This container contains the certificate element configuration.


KeyMCertificate: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+parameter
KeyMCertificateId: EcucIntegerParamDef
min = 0
max = 65535
symbolicNameValue = true
+parameter
KeyMCertificateMaxLength: EcucIntegerParamDef
min = 1
max = 65535
+parameter
KeyMCertificateName: EcucStringParamDef
+parameter
KeyMCertificateVerifyCallbackNotificationFunc: EcucFunctionNameDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
KeyMServiceCertificateCallbackNotificationFunc: EcucFunctionNameDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
KeyMCertUpperHierarchicalCertRef: EcucReferenceDef
+destination
+parameter
KeyMCertFormatType: EcucEnumerationParamDef
+literal
X509: EcucEnumerationLiteralDef
+literal
CVC: EcucEnumerationLiteralDef
+literal
CRL: EcucEnumerationLiteralDef
+parameter
KeyMCertAlgorithmType: EcucEnumerationParamDef
+literal
RSA: EcucEnumerationLiteralDef
+literal
ECC: EcucEnumerationLiteralDef
+reference
KeyMCertCsmSignatureVerifyJobRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
CsmJob: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
(from Csm)
+reference
KeyMCertCsmSignatureVerifyKeyRef: EcucReferenceDef
requiresSymbolicNameValue = true
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
CsmKey: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
(from Csm)
+reference
KeyMCertTimebaseRef: EcucReferenceDef
requiresSymbolicNameValue = true
lowerMultiplicity = 1
upperMultiplicity = 1
+destination
StbMSynchronizedTimeBase: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
(from StbMSynchronizedTimeBase)
+reference
KeyMCertPrivateKeyStorageCryptoKeyRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
KeyMCryptoKey: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+reference
KeyMCertCertificateElementRuleRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+destination
KeyMCertificateElementRule: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+subContainer
KeyMCertificateElement: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+parameter
KeyMCertificateElementId: EcucIntegerParamDef
min = 0
max = 65535
symbolicNameValue = true
+parameter
KeyMCertificateElementObjectId: EcucStringParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
KeyMCertificateElementOfStructure: EcucEnumerationParamDef
+literal
CertificateVersionNumber: EcucEnumerationLiteralDef
+literal
CertificateSerialNumber: EcucEnumerationLiteralDef
+literal
CertificateSignatureAlgorithmID: EcucEnumerationLiteralDef
+literal
CertificateIssuerName: EcucEnumerationLiteralDef
+literal
CertificateValidityPeriodNotBefore: EcucEnumerationLiteralDef
+literal
CertificateValidityPeriodNotBefore: EcucEnumerationLiteralDef
+literal
CertificateValidityPeriodNotAfter: EcucEnumerationLiteralDef
+literal
CertificateSubjectName: EcucEnumerationLiteralDef
+literal
CertificateSubjectPublicKeyInfo_PublicKeyAlgorithm: EcucEnumerationLiteralDef
+literal
CertificateSubjectPublicKeyInfo_SubjectPublicKey: EcucEnumerationLiteralDef
+literal
CertificateIssuerUniqueIdentifier: EcucEnumerationLiteralDef
+literal
CertificateSubjectUniqueIdentifier: EcucEnumerationLiteralDef
+literal
CertificateExtension: EcucEnumerationLiteralDef
+literal
CertificateSignatureAlgorithm: EcucEnumerationLiteralDef
+literal
CertificateSignature: EcucEnumerationLiteralDef
+parameter
KeyMCertificateElementMaxLength: EcucIntegerParamDef
min = 1
max = 65535
+parameter
KeyMCertificateElementHasIteration: EcucBooleanParamDef
defaultValue = false
+parameter
KeyMCertificateElementObjectType: EcucIntegerParamDef
min = 0
max = 255
lowerMultiplicity = 0
upperMultiplicity = 1

10.1.4 KeyMCertificateElement

SWS Item	**ECUC_KeyM_00035 :**
Container Name	KeyMCertificateElement
Parent Container	KeyMCertificate
Description	This container contains the certificate element configuration.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00040 :**		
Name	KeyMCertificateElementHasIteration		
Parent Container	KeyMCertificateElement		
Description	Defines if the certificate element can occur more than one time. If so, the iterator can be used to retrieve the individual data values of this certificate element.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00036 :**		
Name	KeyMCertificateElementId		
Parent Container	KeyMCertificateElement		
Description	Identifier of a certificate element.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00039 :**		
Name	KeyMCertificateElementMaxLength		
Parent Container	KeyMCertificateElement		
Description	Maximum length in bytes		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00037 :**
Name	KeyMCertificateElementObjectId
Parent Container	KeyMCertificateElement

Description	This is the object identifier (OID) that is used to identify the certificate element within its element structure.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00041 :**		
Name	KeyMCertificateElementObjectType		
Parent Container	KeyMCertificateElement		
Description	Certificate elements are stored in ASN.1 format. In this item the type of ASN.1 TLV can be specified (e.g. INTEGER has the value '2'). This can be used to identify only such certificate elements. If the type is different, the element is not included in the search. If KeyMCertificateElementObjectType is not specified, any ASN.1 encoding datatype is used to read the value.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00038 :**	
Name	KeyMCertificateElementOfStructure	
Parent Container	KeyMCertificateElement	
Description	This defines in which structure the certificate element is located.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CertificateExtension	--
	CertificateIssuerName	--
	CertificateIssuerUniqueIdentifier	--
	CertificateSerialNumber	--
	CertificateSignature	--
	CertificateSignatureAlgorithm	--
	CertificateSignatureAlgorithmID	--
	CertificateSubjectAuthorization	--
	CertificateSubjectName	--
	CertificateSubjectPublicKeyInfo_-PublicKeyAlgorithm	--

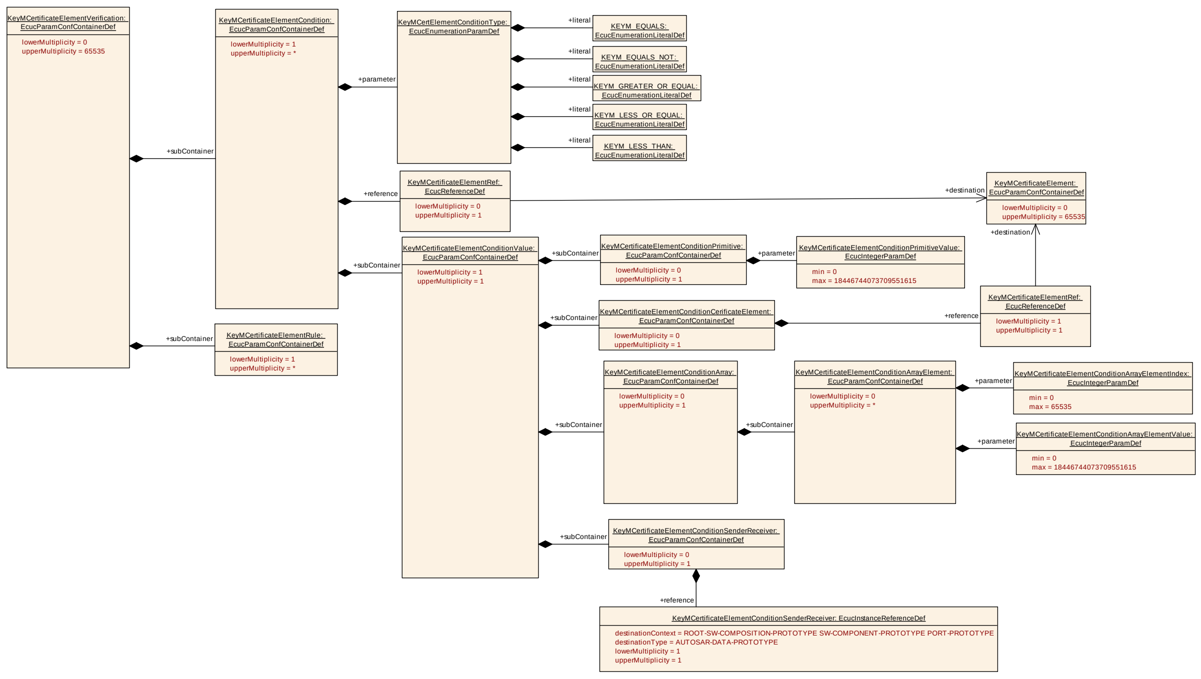
	CertificateSubjectPublicKeyInfo_-SubjectPublicKey	--	
	CertificateSubjectUniqueIdentifier	--	
	CertificateValidityPeriodNotAfter	--	
	CertificateValidityPeriodNotBefore	--	
	CertificateVersionNumber	--	
	RevokedCertificates	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.5 KeyMCertificateElementVerification

SWS Item	**ECUC_KeyM_00004 :**
Container Name	KeyMCertificateElementVerification
Parent Container	KeyM
Description	This container defines if and how certificate elements are to be verified.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
KeyMCertificateElementCondition	1..*	This container contains the configuration of KeyElement compare conditions which can be used as arguments for a KeyMCertificateElementRule. One KeyMCertificateElementCondition shall contain either one KeyMCertificateElementSwcCallback or one KeyMCertificateElementSwcSRDataElementRef or one KeyMCertificateElementSwcSRDataElementValueRef.
KeyMCertificateElementRule	1..*	This container contains the configuration of a mode rule which represents a logical expression with KeyMCertificateElementCondition or other KeyMCertificateElementRule as arguments. All arguments are processed with the operator defined by KeyMLogicalOperator, for instance: Argument_A AND Argument_B AND Argument_C.


KeyMCertificateElementVerification: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+subContainer
KeyMCertificateElementCondition: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+parameter
KeyMCertElementConditionType: EcucEnumerationParamDef
+literal
KEYM_EQUALS: EcucEnumerationLiteralDef
+literal
KEYM_EQUALS_NOT: EcucEnumerationLiteralDef
+literal
KEYM_GREATER_OR_EQUAL: EcucEnumerationLiteralDef
+literal
KEYM_LESS_OR_EQUAL: EcucEnumerationLiteralDef
+literal
KEYM_LESS_THAN: EcucEnumerationLiteralDef
+reference
KeyMCertificateElementRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
KeyMCertificateElement: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+destination
+subContainer
KeyMCertificateElementConditionValue: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
KeyMCertificateElementConditionPrimitive: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
KeyMCertificateElementConditionPrimitiveValue: EcucIntegerParamDef
min = 0
max = 18446744073709551615
+subContainer
KeyMCertificateElementConditionCerificateElement: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
KeyMCertificateElementRef: EcucReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
KeyMCertificateElementConditionArray: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
KeyMCertificateElementConditionArrayElement: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
KeyMCertificateElementConditionArrayElementIndex: EcucIntegerParamDef
min = 0
max = 65535
+parameter
KeyMCertificateElementConditionArrayElementValue: EcucIntegerParamDef
min = 0
max = 18446744073709551615
+subContainer
KeyMCertificateElementConditionSenderReceiver: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
KeyMCertificateElementConditionSenderReceiver: EcucInstanceReferenceDef
destinationContext = ROOT-SW-COMPOSITION-PROTOTYPE SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE
destinationType = AUTOSAR-DATA-PROTOTYPE
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
KeyMCertificateElementRule: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *

10.1.6 KeyMCertificateElementRule

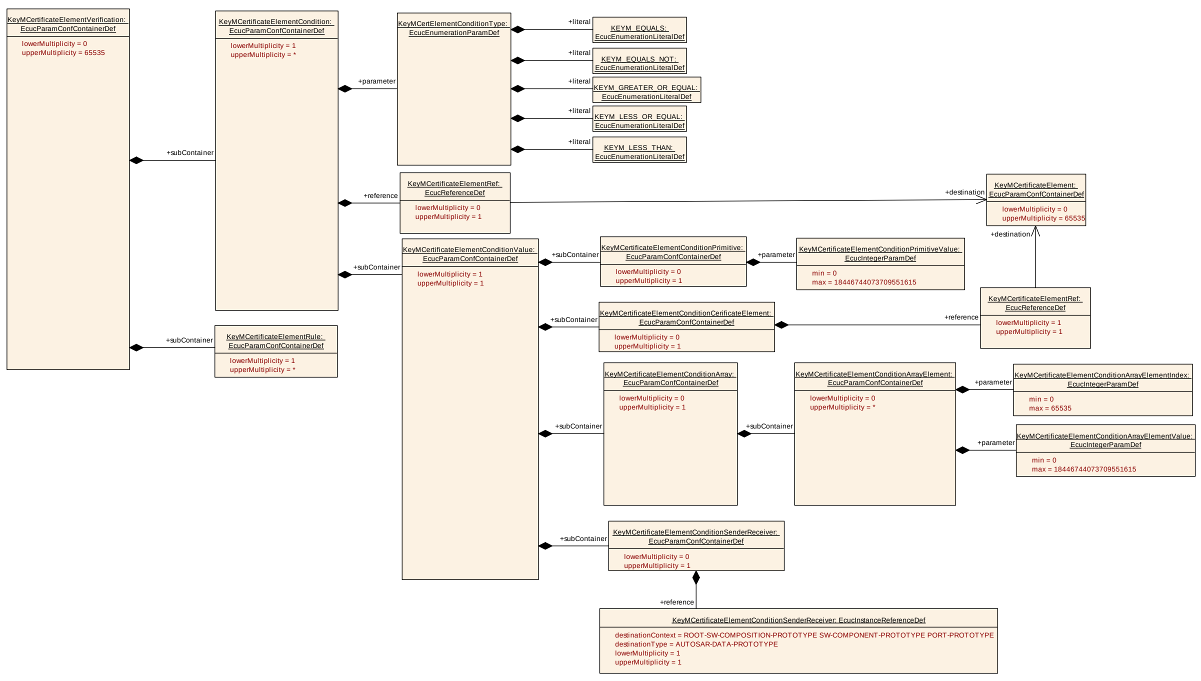
SWS Item	**ECUC_KeyM_00043 :**
Container Name	KeyMCertificateElementRule
Parent Container	KeyMCertificateElementVerification
Description	This container contains the configuration of a mode rule which represents a logical expression with KeyMCertificateElementCondition or other KeyMCertificateElementRule as arguments. All arguments are processed with the operator defined by KeyMLogicalOperator, for instance: Argument_A AND Argument_B AND Argument_C.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00057 :**		
Name	KeyMLogicalOperator		
Parent Container	KeyMCertificateElementRule		
Description	This parameter specifies the logical operator to be used in the logical expression. If the expression only consists of a single condition this parameter shall not be used.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	KEYM_AND	--	
	KEYM_OR	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00058 :**
Name	KeyMArgumentRef

Parent Container	KeyMCertificateElementRule		
Description	This is a choice reference either to a condition or another rule serving as sub-expression.		
Multiplicity	1..*		
Type	Choice reference to [KeyMCertificateElementCondition , KeyMCertificateElementRule]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers


KeyMCertificateElementVerification: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+subContainer
KeyMCertificateElementCondition: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+parameter
KeyMCertElementConditionType: EcucEnumerationParamDef
+literal
KEYM_EQUALS: EcucEnumerationLiteralDef
+literal
KEYM_EQUALS_NOT: EcucEnumerationLiteralDef
+literal
KEYM_GREATER_OR_EQUAL: EcucEnumerationLiteralDef
+literal
KEYM_LESS_OR_EQUAL: EcucEnumerationLiteralDef
+literal
KEYM_LESS_THAN: EcucEnumerationLiteralDef
+reference
KeyMCertificateElementRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
KeyMCertificateElement: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+destination
+subContainer
KeyMCertificateElementConditionValue: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
KeyMCertificateElementConditionPrimitive: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
KeyMCertificateElementConditionPrimitiveValue: EcucIntegerParamDef
min = 0
max = 18446744073709551615
+subContainer
KeyMCertificateElementConditionCerificateElement: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
KeyMCertificateElementRef: EcucReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
KeyMCertificateElementRule: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+subContainer
KeyMCertificateElementConditionArray: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
KeyMCertificateElementConditionArrayElement: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
KeyMCertificateElementConditionArrayElementIndex: EcucIntegerParamDef
min = 0
max = 65535
+parameter
KeyMCertificateElementConditionArrayElementValue: EcucIntegerParamDef
min = 0
max = 18446744073709551615
+subContainer
KeyMCertificateElementConditionSenderReceiver: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
KeyMCertificateElementConditionSenderReceiver: EcucInstanceReferenceDef
destinationContext = ROOT-SW-COMPOSITION-PROTOTYPE SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE
destinationType = AUTOSAR-DATA-PROTOTYPE
lowerMultiplicity = 1
upperMultiplicity = 1

10.1.7 KeyMCertificateElementCondition

SWS Item	**ECUC_KeyM_00042 :**
Container Name	KeyMCertificateElementCondition
Parent Container	KeyMCertificateElementVerification
Description	This container contains the configuration of KeyElement compare conditions which can be used as arguments for a KeyMCertificateElementRule. One KeyMCertificateElementCondition shall contain either one KeyMCertificateElementSwcCallback or one KeyMCertificateElementSwcSRDataElementRef or one KeyMCertificateElementSwcSRDataElementValueRef.

Configuration Parameters

SWS Item	**ECUC_KeyM_00044 :**		
Name	KeyMCertElementConditionType		
Parent Container	KeyMCertificateElementCondition		
Description	This parameter specifies what kind of comparison that is made for the evaluation of the mode condition.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	KEYM_EQUALS	--	
	KEYM_EQUALS_NOT	--	
	KEYM_GREATER_OR_EQUAL	--	
	KEYM_LESS_OR_EQUAL	--	
	KEYM_LESS_THAN	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00045 :**		
Name	KeyMCertificateElementRef		
Parent Container	KeyMCertificateElementCondition		
Description	Reference to a certificate element used for the condition.		
Multiplicity	0..1		
Type	Reference to [KeyMCertificateElement]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
KeyMCertificateElementConditionValue	1	This container contains the configuration of a compare value.

10.1.8 KeyMCertificateElementConditionPrimitive

SWS Item	**ECUC_KeyM_00047 :**
Container Name	KeyMCertificateElementConditionPrimitive
Parent Container	KeyMCertificateElementConditionValue
Description	This container contains the configuration of a primitive compare value.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00053 :**		
Name	KeyMCertificateElementConditionPrimitiveValue		
Parent Container	KeyMCertificateElementConditionPrimitive		
Description	Primitive compare value		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.9 KeyMCertificateElementConditionArray

SWS Item	**ECUC_KeyM_00048 :**
Container Name	KeyMCertificateElementConditionArray
Parent Container	KeyMCertificateElementConditionValue
Description	This container contains the configuration of a array compare value.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
KeyMCertificateElementConditionArrayElement	0..*	This container contains the configuration of a array compare value.

10.1.10 KeyMCertificateElementConditionArrayElement

SWS Item	**ECUC_KeyM_00054 :**
Container Name	KeyMCertificateElementConditionArrayElement
Parent Container	KeyMCertificateElementConditionArray
Description	This container contains the configuration of a array compare value.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00055 :**		
Name	KeyMCertificateElementConditionArrayElementIndex		
Parent Container	KeyMCertificateElementConditionArrayElement		
Description	Index to an element of the compare value array.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00056 :**		
Name	KeyMCertificateElementConditionArrayElementValue		
Parent Container	KeyMCertificateElementConditionArrayElement		
Description	Value of an array element compare value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.11 KeyMCertificateElementConditionCerificateElement

SWS Item	**ECUC_KeyM_00049 :**
Container Name	KeyMCertificateElementConditionCerificateElement
Parent Container	KeyMCertificateElementConditionValue
Description	This container contains the configuration of a certificate element as a compare value.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00051 :**		
Name	KeyMCertificateElementRef		
Parent Container	KeyMCertificateElementConditionCerificateElement		
Description	Reference to another certificate element.		
Multiplicity	1		
Type	Reference to [KeyMCertificateElement]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.12 KeyMCertificateElementConditionValue

SWS Item	**ECUC_KeyM_00046 :**
Container Name	KeyMCertificateElementConditionValue
Parent Container	KeyMCertificateElementCondition
Description	This container contains the configuration of a compare value.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
KeyMCertificateElementConditionArray	0..1	This container contains the configuration of a array compare value.
KeyMCertificateElementConditionCerificateElemen t	0..1	This container contains the configuration of a certificate element as a compare value.
KeyMCertificateElementConditionPrimitive	0..1	This container contains the configuration of a primitive compare value.
KeyMCertificateElementConditionSenderReceiver	0..1	This container contains the configuration of a dynamic compare value in a sender-/receiver interface.

10.1.13 KeyMCertificateElementConditionSenderReceiver

SWS Item	**ECUC_KeyM_00050 :**
Container Name	KeyMCertificateElementConditionSenderReceiver
Parent Container	KeyMCertificateElementConditionValue
Description	This container contains the configuration of a dynamic compare value in a sender-/receiver interface.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00052 :**		
Name	KeyMCertificateElementConditionSenderReceiver		
Parent Container	KeyMCertificateElementConditionSenderReceiver		
Description	This parameter references a mode in a particular mode request port of a software component that is used for the condition.		
Multiplicity	1		
Type	Instance reference to [AUTOSAR-DATA-PROTOTYPE context: ROOT-SW-COMPOSITION-PROTOTYPE SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.14 KeyMCryptoKey

SWS Item	**ECUC_KeyM_00005 :**
Container Name	KeyMCryptoKey
Parent Container	KeyM
Description	This container contains the crypto keys that can be updated.
Configuration Parameters	

SWS Item	**ECUC_KeyM_00067 :**
Name	KeyMCryptoCsmVerifyJobType

Parent Container	KeyMCryptoKey		
Description	Specifies what type of function for key verification operation is used.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	KEYM_VERIFY_AEADDECRYPT	--	
	KEYM_VERIFY_AEADENCRYPT	--	
	KEYM_VERIFY_DECRYPT	--	
	KEYM_VERIFY_ENCRYPT	--	
	KEYM_VERIFY_MACGENERATE	--	
	KEYM_VERIFY_MACVERIFY	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter is only needed if KeymGeneral/KeyMCryptoKey/KeyMCryptoKeyVerifyFunctionEnabled is set to TRUE.		

SWS Item	**ECUC_KeyM_00069 :**		
Name	KeyMCryptoKeyCryptoProps		
Parent Container	KeyMCryptoKey		
Description	If set, it will provide additional hints to the crypto key that is used by KeyM to identify the key. Typical approach is to set the value to the SHE-Slot ID where the key was placed to. If present, the KeyM will take the information and identify the key by its slot ID. The slot information will be extracted from the corresponding field of the M1M2M3 data.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00068 :**
Name	KeyMCryptoKeyGenerationInfo
Parent Container	KeyMCryptoKey
Description	This data may contain static data for key derivation. If a key is configured to be derived from another key and this configuration item is set, the data will be added as salt.
Multiplicity	0..1
Type	EcucStringParamDef
Default value	--
maxLength	--

minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00061 :**		
Name	KeyMCryptoKeyGenerationType		
Parent Container	KeyMCryptoKey		
Description	Specifies how the CryptoKey will be generated. If it is derived from another key or simply stored with KeyElementSet.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	KEYM_DERIVED_KEY	--	
	KEYM_STORED_KEY	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00059 :**		
Name	KeyMCryptoKeyId		
Parent Container	KeyMCryptoKey		
Description	Identifier of the crypto key. The set of configured identifiers shall be consecutive and gapless.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00060 :**		
Name	KeyMCryptoKeyMaxLength		
Parent Container	KeyMCryptoKey		
Description	The maximum size in bytes of a CryptoKey.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00062 :**		
Name	KeyMCryptoKeyName		
Parent Container	KeyMCryptoKey		
Description	Provides a unique name of the key for identification. The key master will reference keys by this unique key name.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00063 :**		
Name	KeyMCryptoKeyStorage		
Parent Container	KeyMCryptoKey		
Description	Specify the storage location of the certificate.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	KEYM_STORAGE_IN_CSM	--	
	KEYM_STORAGE_IN_NVM	--	
	KEYM_STORAGE_IN_RAM	--	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

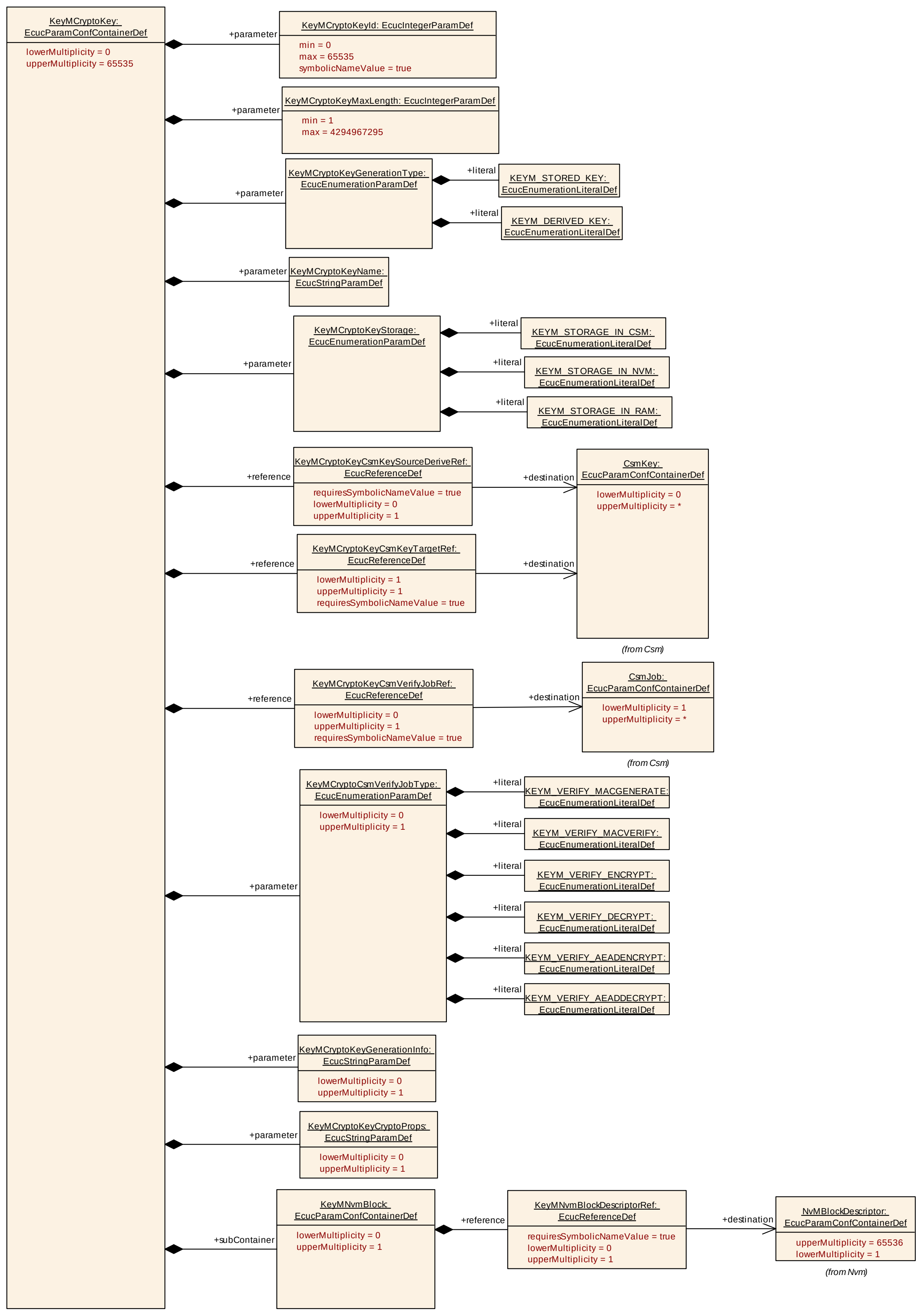
SWS Item	**ECUC_KeyM_00064 :**		
Name	KeyMCryptoKeyCsmKeySourceDeriveRef		
Parent Container	KeyMCryptoKey		
Description	Defines a reference to the associated CSM key that is used as source for the key derivation of this key.		
Multiplicity	0..1		
Type	Symbolic name reference to [CsmKey]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: Only needed if KeyMCryptoKeyGenerationType is set to KEYM_DERIVED_KEY		

SWS Item	**ECUC_KeyM_00065 :**		
Name	KeyMCryptoKeyCsmKeyTargetRef		
Parent Container	KeyMCryptoKey		
Description	Defines a reference to the associated CSM key that shall be generated.		
Multiplicity	1		
Type	Symbolic name reference to [CsmKey]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: Only needed if KeyMCryptoKeyGenerationType is set to KEYM_DERIVED_KEY		

SWS Item	**ECUC_KeyM_00066 :**		
Name	KeyMCryptoKeyCsmVerifyJobRef		
Parent Container	KeyMCryptoKey		
Description	Defines the crypto job that the key verify function can use for verification of a certain key.		
Multiplicity	0..1		
Type	Symbolic name reference to [CsmJob]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00076 :**		
Name	KeyMCryptoKeyNvmBlockRef		
Parent Container	KeyMCryptoKey		
Description	Defines a reference to the NvM block where the key is going to be stored.		
Multiplicity	0..1		
Type	Reference to [KeyMNvmBlock]		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: Only necessary if KeyMCryptoKeyStorage is set to KEYM_STORAGE_IN_NVM		

No Included Containers


KeyMCryptoKey: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+parameter
KeyMCryptoKeyId: EcucIntegerParamDef
min = 0
max = 65535
symbolicNameValue = true
+parameter
KeyMCryptoKeyMaxLength: EcucIntegerParamDef
min = 1
max = 4294967295
+parameter
KeyMCryptoKeyGenerationType: EcucEnumerationParamDef
+literal
KEYM_STORED_KEY: EcucEnumerationLiteralDef
+literal
KEYM_DERIVED_KEY: EcucEnumerationLiteralDef
+parameter
KeyMCryptoKeyName: EcucStringParamDef
+parameter
KeyMCryptoKeyStorage: EcucEnumerationParamDef
+literal
KEYM_STORAGE_IN_CSM: EcucEnumerationLiteralDef
+literal
KEYM_STORAGE_IN_NVM: EcucEnumerationLiteralDef
+literal
KEYM_STORAGE_IN_RAM: EcucEnumerationLiteralDef
+reference
KeyMCryptoKeyCsmKeySourceDeriveRef: EcucReferenceDef
requiresSymbolicNameValue = true
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
CsmKey: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
(from Csm)
+reference
KeyMCryptoKeyCsmKeyTargetRef: EcucReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
+reference
KeyMCryptoKeyCsmVerifyJobRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
CsmJob: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
(from Csm)
+parameter
KeyMCryptoCsmVerifyJobType: EcucEnumerationParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+literal
KEYM_VERIFY_MACGENERATE: EcucEnumerationLiteralDef
+literal
KEYM_VERIFY_MACVERIFY: EcucEnumerationLiteralDef
+literal
KEYM_VERIFY_ENCRYPT: EcucEnumerationLiteralDef
+literal
KEYM_VERIFY_DECRYPT: EcucEnumerationLiteralDef
+literal
KEYM_VERIFY_AEADENCRYPT: EcucEnumerationLiteralDef
+literal
KEYM_VERIFY_AEADDECRYPT: EcucEnumerationLiteralDef
+parameter
KeyMCryptoKeyGenerationInfo: EcucStringParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
KeyMCryptoKeyCryptoProps: EcucStringParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
KeyMNvmBlock: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
KeyMNvmBlockDescriptorRef: EcucReferenceDef
requiresSymbolicNameValue = true
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
NvMBlockDescriptor: EcucParamConfContainerDef
upperMultiplicity = 65536
lowerMultiplicity = 1
(from Nvm)

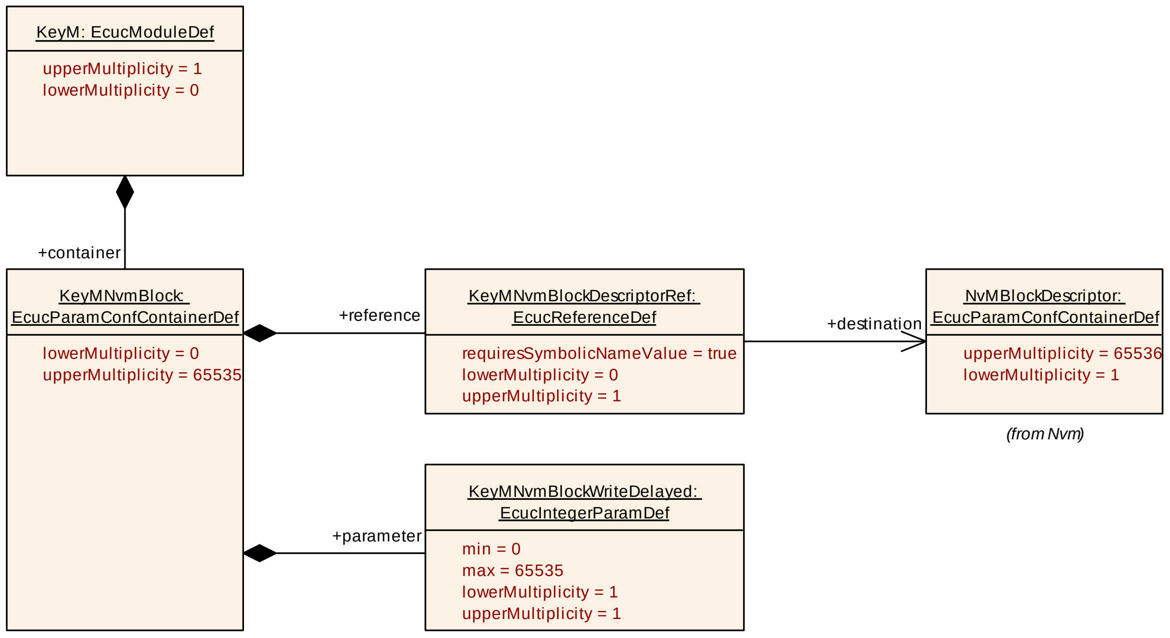
10.1.15 KeyMNvmBlock

SWS Item	**ECUC_KeyM_00070 :**		
Container Name	KeyMNvmBlock		
Parent Container	KeyM		
Description	Configuration of optional usage of Nvm in case the KeyM module requires non volatile memory in the Ecu to store information (e.g. crypto keys or certificates).		
Configuration Parameters			

SWS Item	**ECUC_KeyM_00072 :**		
Name	KeyMNvmBlockWriteDelayed		
Parent Container	KeyMNvmBlock		
Description	This is the delay time in ms to write a key to NVM after it has been updated. A value of 0 means, that the key is written immediately after it has been updated. If several keys are update that are assigned to the same container, the first delay time expiration shall be used. All keys that have been updated during that time shall be updated and its delay timer shall be stopped.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUC_KeyM_00071 :**		
Name	KeyMNvmBlockDescriptorRef		
Parent Container	KeyMNvmBlock		
Description	Reference to the Nvm block description in the Nvm module configuration.		
Multiplicity	0..1		
Type	Symbolic name reference to [NvMBlockDescriptor]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers



10.1.16 KeyMSecurityEventRefs

SWS Item	**ECUC_KeyM_00079 :**		
Container Name	KeyMSecurityEventRefs		
Parent Container	KeyMGeneral		
Description	Container for the references to IdsMEvent elements representing the security events that the KeyM module shall report to the IdsM in case the coresponding security related event occurs (and if KeyMEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events. **Tags:** atp.Status=draft		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

SWS Item	**ECUC_KeyM_00084 :**
Name	KEYM_SEV_CERT_VERIF_FAILED
Parent Container	KeyMSecurityEventRefs
Description	A request to verify a certificate against a certificate chain was not successful. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to [IdsMEvent]
Post-Build Variant Multiplicity	false
Post-Build Variant Value	false

Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00082 :**		
Name	KEYM_SEV_INST_INTERMEDIATE_CERT_OP		
Parent Container	KeyMSecurityEventRefs		
Description	Attempt to install an intermediate certificate. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Symbolic name reference to [IdsMEvent]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

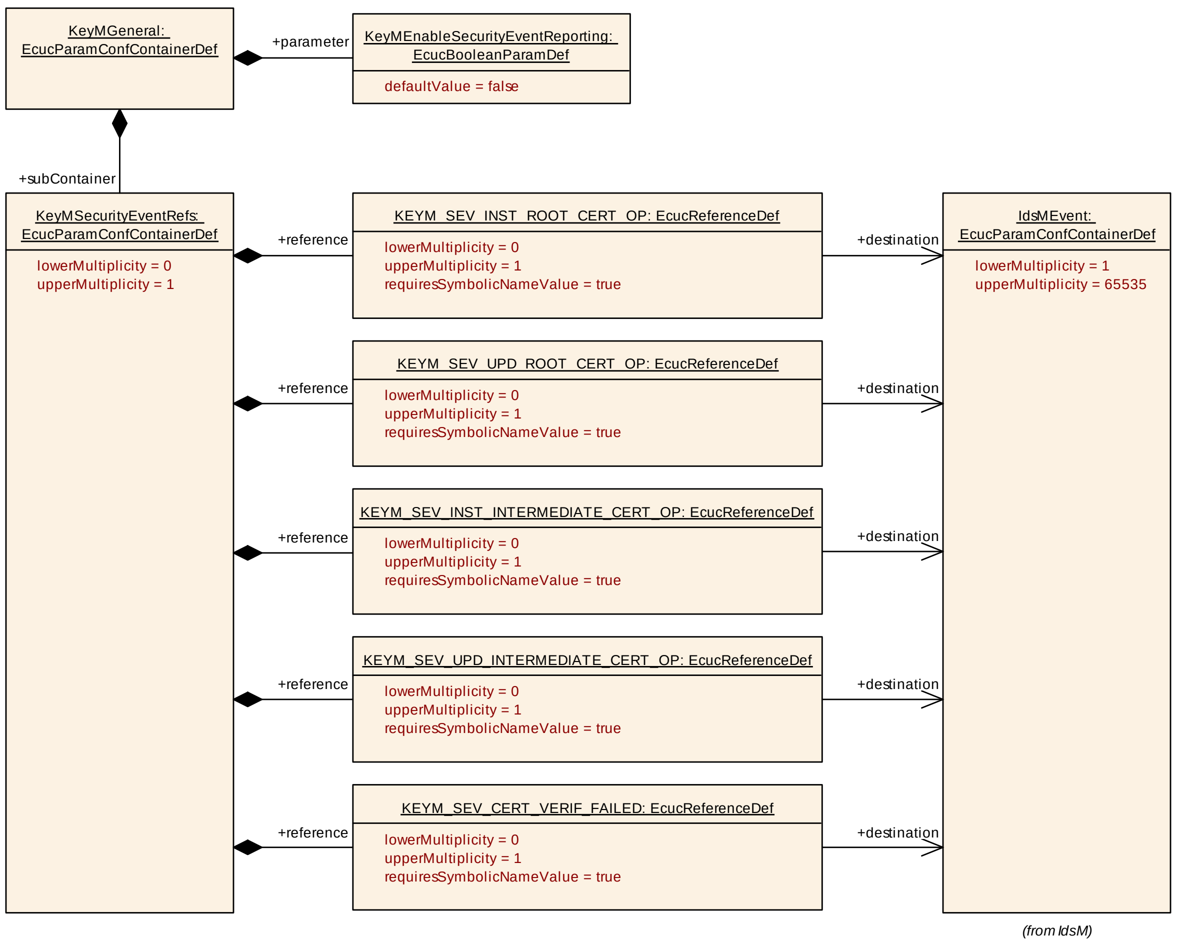
SWS Item	**ECUC_KeyM_00080 :**		
Name	KEYM_SEV_INST_ROOT_CERT_OP		
Parent Container	KeyMSecurityEventRefs		
Description	Attempt to install a root certificate. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Symbolic name reference to [IdsMEvent]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00083 :**
Name	KEYM_SEV_UPD_INTERMEDIATE_CERT_OP
Parent Container	KeyMSecurityEventRefs
Description	Attempt to update an existing intermediate certificate. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to [IdsMEvent]
Post-Build Variant Multiplicity	false

Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_KeyM_00081 :**		
Name	KEYM_SEV_UPD_ROOT_CERT_OP		
Parent Container	KeyMSecurityEventRefs		
Description	Attempt to update an existing root certificate. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Symbolic name reference to [IdsMEvent]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers


KeyMGeneral: EcucParamConfContainerDef
+parameter
KeyMEnableSecurityEventReporting: EcucBooleanParamDef
defaultValue = false
+subContainer
KeyMSecurityEventRefs: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
KEYM_SEV_INST_ROOT_CERT_OP: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
KEYM_SEV_UPD_ROOT_CERT_OP: EcucReferenceDef
KEYM_SEV_INST_INTERMEDIATE_CERT_OP: EcucReferenceDef
KEYM_SEV_UPD_INTERMEDIATE_CERT_OP: EcucReferenceDef
KEYM_SEV_CERT_VERIF_FAILED: EcucReferenceDef
IdsMEvent: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 65535
(from IdsM)

10.2 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

Additional module-specific published parameters are listed below if applicable.

11 Not applicable requirements

[SWS_KeyM_00174]⌈ These requirements are not applicable to this specification.⌋(SRS_CryptoStack_00003, SRS_CryptoStack_00006, SRS_CryptoStack_00008, SRS_CryptoStack_00009, SRS_CryptoStack_00014, SRS_CryptoStack_00015, SRS_CryptoStack_00034, SRS_CryptoStack_00036, SRS_CryptoStack_00075, SRS_CryptoStack_00076, SRS_CryptoStack_00079, SRS_CryptoStack_00081, SRS_CryptoStack_00082, SRS_CryptoStack_00084, SRS_CryptoStack_00088, SRS_CryptoStack_00089, SRS_CryptoStack_00095, SRS_CryptoStack_00096, SRS_CryptoStack_00097, SRS_CryptoStack_00098, SRS_CryptoStack_00102, SRS_CryptoStack_00104, SRS_CryptoStack_00122, SRS_CryptoStack_00123, SRS_CryptoStack_00124)