MICROSAR CRY DRIVER

Technical Reference

DrvCry_Rh850Icus

Version 1.06.01

Authors	Tobias Finke
Status	Released

Document Information

History

Author	Date	Version	Remarks
Tobias Finke	2015-05-22	1.00.00	Initial Version of MICROSAR CRY DRIVER
Tobias Finke	2015-07-28	1.01.00	- Added Sym Key Wrapping Service. - Added prerequisites for the Initialization. - Added Limitation for multiple calls of some Cry_<Primitive>Update() functions.
Tobias Finke	2015-08-20	1.02.00	- Added Limitation: Different Services can't be accessed in parallel. - Added description for aborting services. - Changed include structure.
Tobias Finke	2016-01-11	1.02.01	- Cry_ShePrngGenerate accepts a resultLength smaller than 16 byte.
Tobias Finke	2016-06-17	1.03.00	- Config option if length of mac in MacVerify is interpreted as bits. - Config option how keyIds are mapped. - Config option for ICUS base address. - Change in the way how M4 and M5 are returned after key provisioning.
Tobias Finke	2016-08-01	1.03.01	- Added Timeout-API - Changed default mapping in the mapped Use-Case for RAM_KEY from 0xEE to 0x00.
Tobias Finke	2016-11-24	1.04.00	- Added Configuration with DaVinci Configurator 5 - Removed SymKeyWrapForkeyProvisioning
Tobias Finke	2017-04-06	1.05.00	- Updated function descriptions - Config option for FHVE support - Config option for hardware error code callout - Config option for data flash control callouts - Config option for data flash synchronization callouts. - Description of exclusive areas
Tobias Finke	2017-04-13	1.05.01	- Fixed formatting and spelling
Tobias Finke	2017-07.14	1.05.02	- Adaption to base component
Tobias Finke	2017-10-12	1.06.00	- Added Self Test - SafeBsw Release
Tobias Finke	2017-10-19	1.06.01	- Changed include structure

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_CryptoServiceManager.pdf	1.2.0
[2]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[3]	HIS	SHE - Functional Specification	1.1
[4]	RENESAS	User's Manual: RH850/F1L ICUSB	1.00

Caution
This symbol calls your attention to warnings.

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

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.00	- Initial version – BETA
1.00.01	- Fixed miscellaneous warnings
1.01.00	- Fixed AES enc/dec and CMAC gen/ver when using a key slot. - Fixed wrong byte order when reading and writing from/to ICU.
1.02.00	- Changed SFR access from bit-fields to mask operations - MISRA compliance and improved robustness. - Fixed compiler errors in asynchronous configuration. - Added state machine for SHE Services. - Added additional checks for KeyExtract when provisioning is enabled
1.02.01	- Added check before issuing cancel command to prevent blocking of ECU - Fixed missing defines for CRY state machine - Fixed wrong handling of MAC truncation in CmacAes128Gen and CmacAes128VGen
1.02.02	- Fixed issue with request for random numbers smaller than 16 bytes
1.03.00	- Fixed Support for 20 key slots in ICUSD and ICUSE - Config option for variable base address of ICUS - Configurable interpretation of the KeyId - Use CMD_MAC_VERIFY instead of CMD_MAC_GENERATE in mac verification primitive - Added support for mac length in bit for MacVerify
1.03.01	- Fixed buffer overflow in Cry_30_Rh850Icus_KeyExtract when CsmSymKeyExtractMaxKeySize is smaller than 48 bytes and CRY_30_RH850ICUS_KEYWRAPSYM_FOR_PROVISION is STD_OFF
1.03.02	Added Timeout-API
2.00.00	- FEAT-1995: Support CRY(HW) and CRY(SW) in the same SIP - Fixed ECC-Error on P1M because variable located in wrong memory section - MacVerification causes the ECU to block on P1M derivatives
2.00.01	- Added support for FHVE on derivatives like P1M - Fixed wrong software check for correct AuthId
2.01.00	- Added Callouts for Data Flash Synchronization - Added Callouts to handle FACI interface commands - Added Callouts to provide internal ICUS errors to the application - Fixed state handling of random generate in async configuration
2.01.01	- Removed unnecessary includes - Descriptions for SafeBsw are missing - Disable not used helper functions
2.02.00	- Adaption to base component - Fixed Second keyslot not usable in RAW configuration

Component Version	New Features
	- Timeout location callout provides wrong information
2.02.01	- Fixed Compiler Error: Wrong define for memclass in FUNC() macro of Cry_30_Rh850Icus_RngSeedFinishInternal
2.03.00	- Added Self Test - SafeBsw Release
2.03.01	- Changed include structure

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the MICROSAR module CRY_30_RH850ICUS as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	Pre-Compile	
Vendor ID:	CRY_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CRY_MODULE_ID	255 decimal (according to ref. [2])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

This document describes the functionality and API of the CRY module as a hardware dependent module.

The Cryptographic library module (CRY) offers cryptographic primitives. The CRY module is used by the Crypto Service Manager (CSM).

2.1 Architecture Overview

The following figure shows where the CRY_30_RH850ICUS is located in the AUTOSAR architecture.

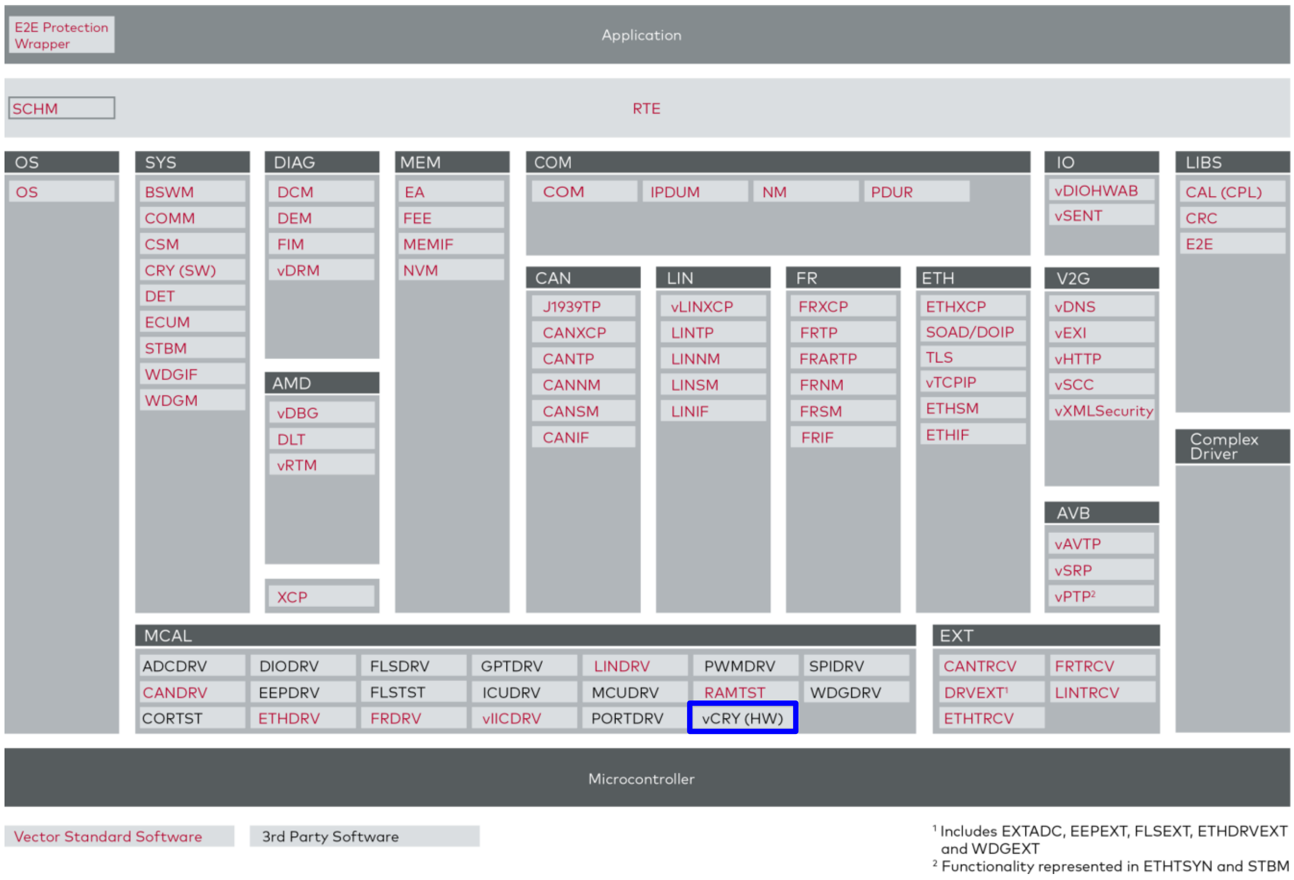

Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the CRY_30_RH850ICUS. These interfaces are described in chapter 5.

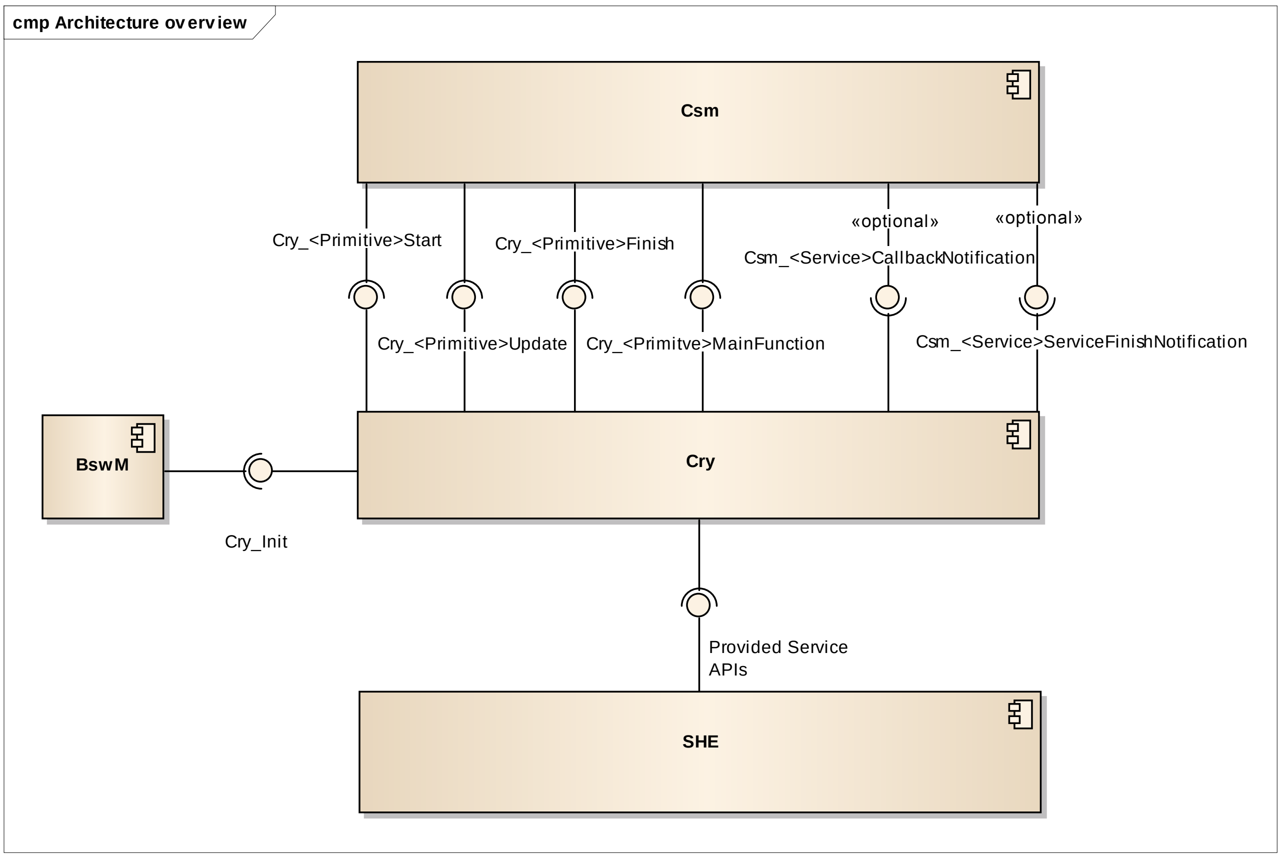

Figure 2-2: Interfaces to adjacent modules of the CRY_30_RH850ICUS

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the CRY_30_RH850ICUS.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Synchronous job processing
Asynchronous job processing
Service for Symmetrical Interface (AES128)
Service for MAC Interface (CMAC)
Service for Random Interface (PRNG)
Service for Symmetrical Key Extract Interface
Service for Symmetrical Key Wrapping Interface

Table 3-1 Supported AUTOSAR standard conform features

3.2 Initialization

Before calling any other functionality of the CRY module the initialization function `Cry_30_Rh850Icus_Init()` has to be called at startup, e.g. by BswM or EcuM.

For API details refer to chapter 5.4.1, ‘Cry_30_Rh850Icus_Init’.

The CRY module assumes that some variables are initialized with certain values at start-up. As not all embedded targets support the initialization of RAM within the start-up code the CRY module provides the function `Cry_30_Rh850Icus_InitMemory()`. This function has to be called during start-up and before `Cry_30_Rh850Icus_Init()` is called.

For API details refer to chapter 5.4.2 ‘Cry_30_Rh850Icus_InitMemory’.

!

Caution
The Renesas Data Flash Access Library must be initialized before any usage of the key storage in the SHE. This is mandatory for enabling the SHE to store keys in the key slots which are located in the data flash.

3.3 States

Due to the hardware limitations of the SHE, only one service can be handled to the same time. Therefore, a service can only be started, if no other service is already running.

To provide this behavior, the CRY module stores a global state (`Cry_30_Rh850Icus_ServiceState_Type`) which is used in all provided services.

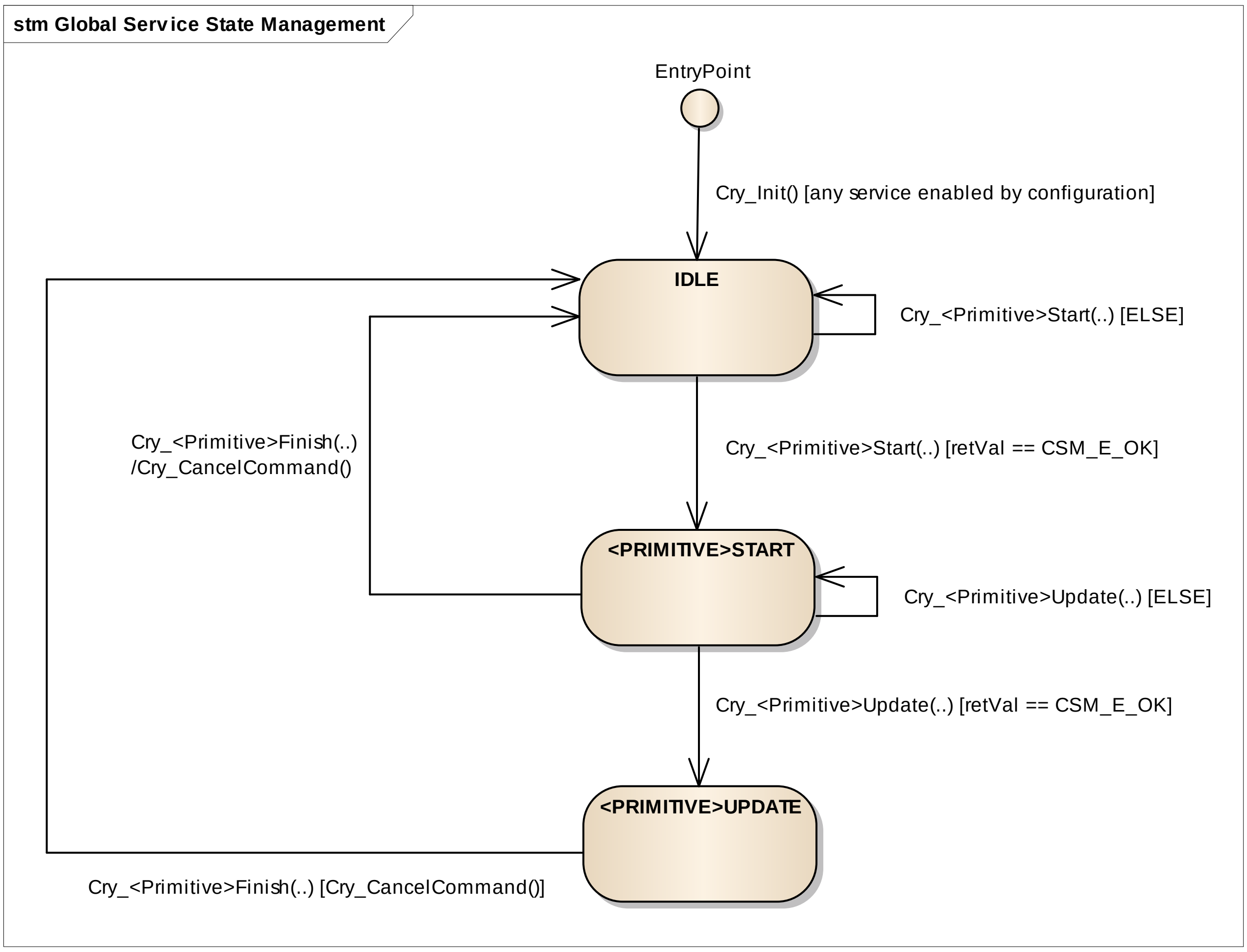
stm Global Service State Management
EntryPoint
Cry_Init() [any service enabled by configuration]
IDLE
Cry_<Primitive>Start(..) [ELSE]
Cry_<Primitive>Finish(..)
/Cry_CancelCommand()
Cry_<Primitive>Start(..) [retVal == CSM_E_OK]
<PRIMITIVE>START
Cry_<Primitive>Update(..) [ELSE]
Cry_<Primitive>Update(..) [retVal == CSM_E_OK]
<PRIMITIVE>UPDATE
Cry_<Primitive>Finish(..) [Cry_CancelCommand()]

Figure 3-1: State chart of the global service state

The global state exists of an IDLE state and a specific START and UPDATE state for each service. This means for example, that there exists an AESDECRYPT128_START state and also an AESENCRYPT128_START state.

The global state machine is initialized by the IDLE state after a successful call of `Cry_30_Rh850Icus_Init()`.

A call of a service specific Start-Function with the return value CSM_E_OK results in a transition from the IDLE state to the service specific START state. An additional call of a Start-Function of any service is rejected by the CRY module, because another service is already started.

If any service is started successfully, a Start-, Update-, or Finish-function call of any other service is always rejected until the global state is reverted to IDLE by the Finish-function of the already started service.

The successful started service can call the service specific Update-Function to pass input data to the service. The function call is only accepted, if the corresponding service was started before successful.

If the Update-Function is called successful, the state machine switches to the service specific UPDATE state. If the Update-Function is called and returned with an error, the state is not changing.

If the Finish-Function from an already started service is called, the global state switches back to IDLE and enables other services to start.

Note
The Update-Function of a service can only be called once due to hardware limitations of the SHE. If the Update-Function is accepted once by the return value CSM_E_OK, there must be no repetition of another Update-call.

Caution
If a service is started, the Finish-Function of this service needs to be called, in order to enable other services to run.

Please refer to chapter 3.9 for more information about the general procedure for a service execution.

3.4 Main Functions

The CRY module implementation provides one main function for each service. When the usage of synchronous job processing is disabled, this main function has to be called cyclically from the CSM main-function context if the corresponding service is active.

For API details refer e.g. to chapter 5.4.7 ‘Cry_30_Rh850Icus_AesEncrypt128MainFunction’.

3.5 Asynchronous Handling

There are some differences in the handling between asynchronous and synchronous mode. When calling a service specific Start, Update or Finish-function, the function stores the parameters in a local buffer and marks the function with an additional state machine for asynchronous function handling in the next main-loop of the corresponding service.

The processing of the data is triggered by the service specific Main-Function. The configured user callback function indicates that the processing is finished carrying the result of the operation. Depending on the result, the next operation can be performed, e.g. a call to the Update-Function.

Caution
All input and output data have to be valid during the whole processing of a service execution. This needs to be guaranteed by the caller of the asynchronous functions.

3.6 Key Handling

The symmetrical keys used by the Cry module are in the format of `Csm_SymKeyType`. This struct consists of a data pointer and length. If the length equals 1, the first aligntype of data represents a keyId which is used to select the key slot of the SHE depending on the configuration parameter `keyIdType`. Otherwise, if the length is 16, the data located at the pointer is loaded as a 128 bit key into the RAM key slot of the SHE. This behavior is handled in the specific start function.

3.7 Key Mapping

Depending on the configuration option CryKeyIdType, the keyId is interpreted in two different ways.

	CRY_KEYIDTYPE_RAW	CRY_KEYIDTYPE_MAPPED
KeyId	**SHE-Keyslot**	
0x00	SECRET_KEY	KEY_RAM
0x01	MASTER_ECU_KEY	KEY_1
0x02	BOOT_MAC_KEY	KEY_2
0x03	BOOT_MAC	KEY_3
0x04	KEY_1	KEY_4
0x05	KEY_2	KEY_5
0x06	KEY_3	KEY_6
0x07	KEY_4	KEY_7
0x08	KEY_5	KEY_8
0x09	KEY_6	KEY_9
0x0A	KEY_7	KEY_10
0x0B	KEY_8	KEY_11
0x0C	KEY_9	KEY_12
0x0D	KEY_10	KEY_13

0x0E	KEY_RAM	KEY_14
0x0F	KEY_11	KEY_15
0x10	KEY_12	KEY_16
0x11	KEY_13	KEY_17
0x12	KEY_14	KEY_18
0x13	KEY_15	KEY_19
0x14	KEY_16	KEY_20
0x15	KEY_17	MASTER_ECU_KEY
0x16	KEY_18	-
0x17	KEY_19	-
0x18	KEY_20	-

Table 3-2 Mapping of KeyId to SHE Keyslots

3.8 Key Update

For updating a key slot of the SHE as specified in [3] and [4], the `Cry_30_Rh850Icus_KeyExtract` service is used.

The `dataPtr` in `Cry_30_Rh850Icus_KeyExtractUpdate()` points to an array which consists of the concatenation of a keyId (1 byte) and the three messages M1 (16 byte), M2 (32 byte) and M3 (16 byte). If `dataLength` is 65, these three messages are written to the SHE. The SHE updates the key slot with the key data. Information like Slot ID and the plaintext key data are encoded in the messages M1 to M3.

After writing M1, M2 and M3 to the SHE, the SHE generates M4 and M5 which can be used to verify the key update procedure. In order to retrieve M4 and M5, the keyPtr of `Cry_30_Rh850Icus_KeyExtractFinish()` is used to store the 48 bytes of data. Ensure that `CsmSymKeyExtractMaxKeySize` in the CSM configuration is set to at least 48 Bytes.

3.9 General procedure of service execution

A service/primitive provides in general three interface functions (Note: Signature of functions differs depending on the service).

> `Cry_30_Rh850Icus_<Primitive>Start()`

> `Cry_30_Rh850Icus_<Primitive>Update()`

> `Cry_30_Rh850Icus_<Primitive>Finish()`

To execute a service completely, the interface functions for the services needs to be called as illustrated above.

Caution

Following rules for the call hierarchy of the interface functions must be fulfilled:

1. `Cry_30_Rh850Icus_<Primitive>Update()` must only be called, if `Cry_30_Rh850Icus_<Primitive>Start()` was called before with the return value CSM_E_OK.
2. `Cry_30_Rh850Icus_<Primitive>Finish()` must only be called, if `Cry_30_Rh850Icus_<Primitive>Start()` was called before with the return value CSM_E_OK. A call of `Cry_30_Rh850Icus_<Primitive>Update()` in between is allowed.
3. If `Cry_30_Rh850Icus_<Primitive>Start()` is called successful (CSM_E_OK), a `Cry_30_Rh850Icus_<Primitive>Finish()` must follow.
4. `Cry_30_Rh850Icus_<Primitive>Finish()` must be called only once after a Start- or Update-Call. Even if the result of the Finish-Call is CSM_E_NOT_OK it must not be repeated.

Note: These rules are mandatory for each service of the CRY module.

Figure 3-2 shows in a sequence chart as example the synchronous procedure of the hardware CRY. It focuses on the illustration of the rules mentioned above.

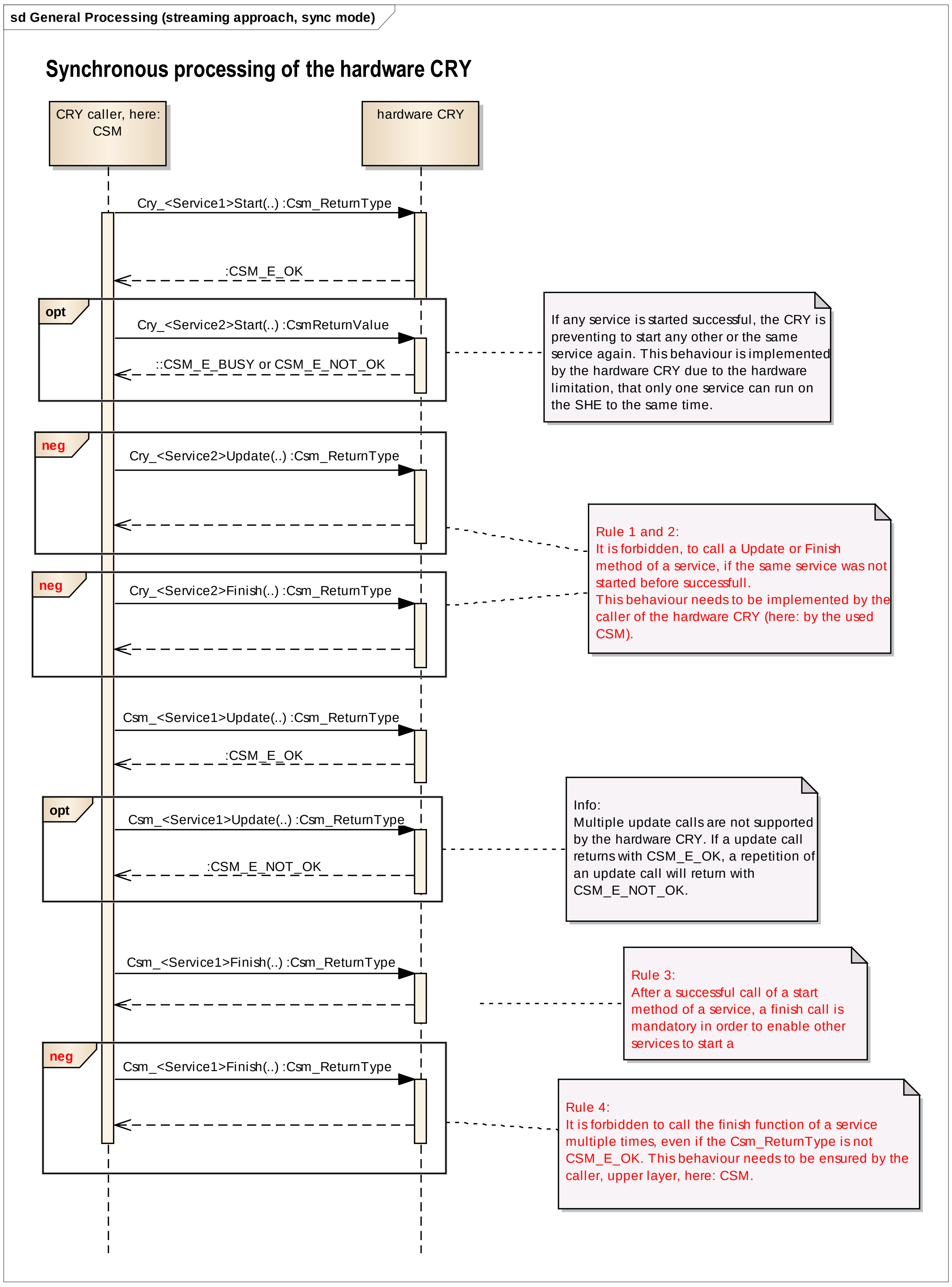
sd General Processing (streaming approach, sync mode)
Synchronous processing of the hardware CRY
CRY caller, here: CSM
hardware CRY
Cry_<Service1>Start(..) :Csm_ReturnType
:CSM_E_OK
opt
Cry_<Service2>Start(..) :CsmReturnValue
::CSM_E_BUSY or CSM_E_NOT_OK
If any service is started successful, the CRY is preventing to start any other or the same service again. This behaviour is implemented by the hardware CRY due to the hardware limitation, that only one service can run on the SHE to the same time.
neg
Cry_<Service2>Update(..) :Csm_ReturnType
neg
Cry_<Service2>Finish(..) :Csm_ReturnType
Rule 1 and 2: It is forbidden, to call a Update or Finish method of a service, if the same service was not started before successfull. This behaviour needs to be implemented by the caller of the hardware CRY (here: by the used CSM).
Csm_<Service1>Update(..) :Csm_ReturnType
:CSM_E_OK
opt
Csm_<Service1>Update(..) :Csm_ReturnType
:CSM_E_NOT_OK
Info: Multiple update calls are not supported by the hardware CRY. If a update call returns with CSM_E_OK, a repetition of an update call will return with CSM_E_NOT_OK.
Csm_<Service1>Finish(..) :Csm_ReturnType
Rule 3: After a successful call of a start method of a service, a finish call is mandatory in order to enable other services to start a
neg
Csm_<Service1>Finish(..) :Csm_ReturnType
Rule 4: It is forbidden to call the finish function of a service multiple times, even if the Csm_ReturnType is not CSM_E_OK. This behaviour needs to be ensured by the caller, upper layer, here: CSM.

Figure 3-2: Sequence chart to show an example of the synchronous processing of the hardware CRY

3.10 Timeout Handling

The Timeout Handling can be done by implementing the two callouts mentioned below. One callout described as LocationCallout is responsible to provide information about the location from where the callout is called. This callout can be used to start a timeout timer in the application. The second callout described as LoopCallout can be used to cancel a running command when it has consumed too much time.

By returning CSM_E_NOT_OK at the LoopCallout, the command CMD_CANCEL will be sent to the SHE to stop the currently executed command. Pay attention to return CSM_E_NOT_OK at all following calls of the LoopCallout with this specific canceled service until it is started again by getting a call of the LocationCallout with the parameter section equal to the value CRY_SHE_TO_SECTION_START_SERVICE. Refer to

Figure 3-3 for an example call sequence of the callouts.

Timeout Handling can be enabled in the configurator. If enabled, the names of the two callout functions have to be defined. Refer to 5.5.1.1 ‘Timeout-API Location Callout’ and 5.5.1.2 ‘Timeout-API Loop Callout’ for more details about the callouts.

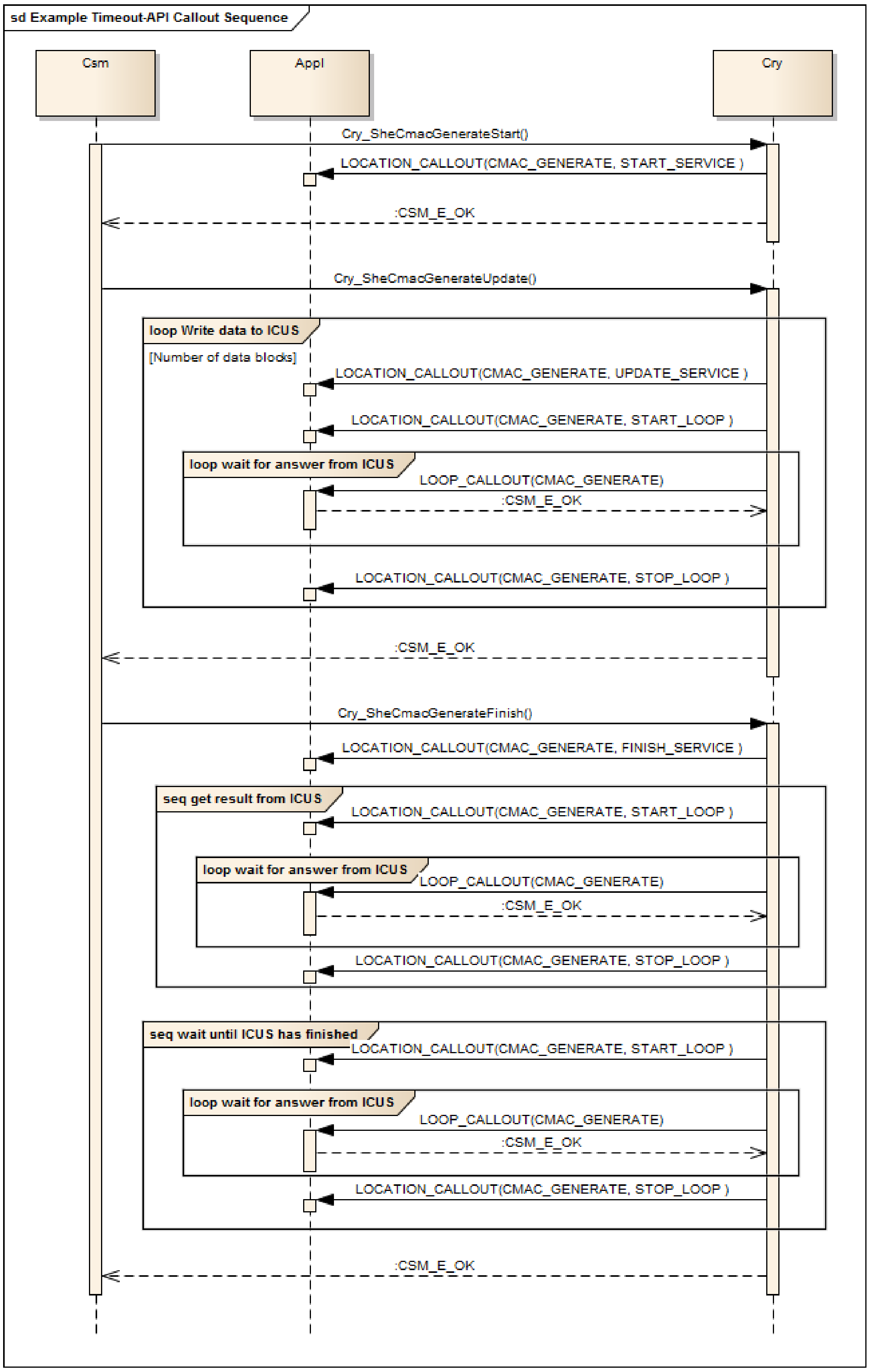
sd Example Timeout-API Callout Sequence
Csm
Appl
Cry
Cry_SheCmacGenerateStart()
LOCATION_CALLOUT(CMAC_GENERATE, START_SERVICE)
:CSM_E_OK
Cry_SheCmacGenerateUpdate()
loop Write data to ICUS
[Number of data blocks]
LOCATION_CALLOUT(CMAC_GENERATE, UPDATE_SERVICE)
LOCATION_CALLOUT(CMAC_GENERATE, START_LOOP)
loop wait for answer from ICUS
LOOP_CALLOUT(CMAC_GENERATE)
:CSM_E_OK
LOCATION_CALLOUT(CMAC_GENERATE, STOP_LOOP)
:CSM_E_OK
Cry_SheCmacGenerateFinish()
LOCATION_CALLOUT(CMAC_GENERATE, FINISH_SERVICE)
seq get result from ICUS
LOCATION_CALLOUT(CMAC_GENERATE, START_LOOP)
loop wait for answer from ICUS
LOOP_CALLOUT(CMAC_GENERATE)
:CSM_E_OK
LOCATION_CALLOUT(CMAC_GENERATE, STOP_LOOP)
seq wait until ICUS has finished
LOCATION_CALLOUT(CMAC_GENERATE, START_LOOP)
loop wait for answer from ICUS
LOOP_CALLOUT(CMAC_GENERATE)
:CSM_E_OK
LOCATION_CALLOUT(CMAC_GENERATE, STOP_LOOP)
:CSM_E_OK

Figure 3-3 Example Sequence for Timeout-API callouts during MAC Generation

3.11 Error Handling

3.11.1 Development Error Reporting

The current implementation of the CRY_30_RH850ICUS module does not report any development errors.

However, the current implementation provides development error detection. Therefore, the pre-processor switch needs to be enabled (CRY_30_RH850ICUS_DEV_ERROR_DETECT == STD_ON). If the switch is enabled, the interface functions execute some plausibility checks of the input parameters (e.g. range check, NULL_PTR check,). This switch should be enabled by default for safety reasons.

3.11.2 Production Code Error Reporting

The current implementation of the CRY_30_RH850ICUS module does not report any production errors.

3.12 Self Test

The Driver provides the functionality to perform a self-test of the CMAC verification functionality.

Test vectors will be used to do a failed and passed verification.

If the self-test fails, the CRY will not succeed to initialize itself.

The self-test can be cyclically called be the user. If the self-test fails with CSM_E_NOT_OK, the user must take apropriate measures.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR CRY_30_RH850ICUS into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the CRY_30_RH850ICUS contains the files which are described in the chapters 4.1.1 and 4.1.2.

4.1.1 Static Files

File Name	Source Code Delivery	Library Delivery	Description
Cry_30_Rh850Icus.c	■		Source file of the Cry_30_Rh850Icus.
Cry_30_Rh850Icus.h	■		Header file of the Cry_30_Rh850Icus.
Cry_30_Rh850Icus_AesDecrypt128.c	■		Source file of the service Cry_30_Rh850Icus_AesDecrypt128.
Cry_30_Rh850Icus_AesDecrypt128.h	■		Header file of the service Cry_30_Rh850Icus_AesDecrypt128.
Cry_30_Rh850Icus_AesEncrypt128.c	■		Source file of the service Cry_30_Rh850Icus_AesEncrypt128.
Cry_30_Rh850Icus_AesEncrypt128.h	■		Header file of the service Cry_30_Rh850Icus_AesEncrypt128.
Cry_30_Rh850Icus_Rng.c	■		Source file of the service Cry_30_Rh850Icus_Rng.
Cry_30_Rh850Icus_Rng.h	■		Header file of the service Cry_30_Rh850Icus_Rng.
Cry_30_Rh850Icus_CmacAes128Gen.c	■		Source file of the service Cry_30_Rh850Icus_CmacAes128Gen.
Cry_30_Rh850Icus_CmacAes128Gen.h	■		Header file of the Cry_30_Rh850Icus_CmacAes128Gen.
Cry_30_Rh850Icus_CmacAes128Ver.c	■		Source file of the service Cry_30_Rh850Icus_CmacAes128Ver.
Cry_30_Rh850Icus_CmacAes128Ver.h	■		Header file of the service Cry_30_Rh850Icus_CmacAes128Ver.

File Name	Source Code Delivery	Library Delivery	Description
Cry_30_Rh850Icus_KeyExtract.c	■		Source file of the service Cry_30_Rh850Icus_KeyExtract.
Cry_30_Rh850Icus_KeyExtract.h	■		Header file of the service Cry_30_Rh850Icus_KeyExtract.
Cry_30_Rh850Icus_KeyWrapSym.c	■		Source file of the service Cry_30_Rh850Icus_KeyWrapSym.
Cry_30_Rh850Icus_KeyWrapSym.h	■		Header file of the service Cry_30_Rh850Icus_KeyWrapSym.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated with the help of Cfg5.

File Name	Description
Cry_30_Rh850Icus_Cfg.h	Contains the configuration of the Cry_30_Rh850Icus.
Cry_30_Rh850Icus_Cfg.c	Contains the generated configuration data of the Cry_30_Rh850Icus.

Table 4-2 Generated files

4.1.3 Callout Files

The component provides a template for the callout functions. The implementation is not fulfilled and needs to be adapted by the user corresponding to his requirements.

File Name	Description
_Cry_30_Rh850Icus_Callouts.h	Contains the declaration of the callout functions which can be activated by pre-processor switches.
_Cry_30_Rh850Icus_Callouts.c	Contains the implementation of the callout functions which can be activated by pre-processor switches. The implementation is not finished. It's just a template with an incomplete example implementation which needs to be adapted.

Table 4-3 Generated files

Edit
Files of chapter 4.1.3 are only templates and must be edited and renamed by the user, if the corresponding configuration switch is enabled.

4.2 Critical Sections

The CRY module calls the following function when entering a critical section:

> void SchM_Enter_Cry_30_Rh850Icus_CRY_30_RH850ICUS _EXCLUSIVE_AREA(void)

When the critical section is left the following function is called by the CRY module:

> void SchM_Exit_Cry_30_Rh850Icus_CRY_30_RH850ICUS _EXCLUSIVE_AREA(void)

These calls are necessary to avoid race conditions of the internal state machines for the synchronous and asynchronous function handling of the different services.

Each call of a Cry_30_Rh850Icus_<Primitive>Start(...) function verifies at the start, if there is no other service is running right now (see Figure 3-1). If no other service is running, the state machine switches to START. From then, it is not possible for other services to start also, because their verification of the IDLE state in the Cry_30_Rh850Icus_<Primitive>Start(…) function fails. The critical section is needed between the verification of the IDLE state and setting the state to START to exclude the risk, that two services verify the state one by one (e.g. in the context of a task switch), before setting it to START.

The length and runtime of the critical section (between Enter and Exit) is short, because it covers the comparison of a state and the setting of a few variables.

4.3 Include Structure

Figure 4-1 shows the include structure of the Cry. All files starting with the Prefix 'Cry' are mapped to files named Cry_30_Rh850Icus. All files which include the word <Primitive> are mapped to n service files. Each service has its own source and header file.

The different primitives are listed in chapter 4.1.1.

Example: Cry_<Primitive>.c is mapped to Cry_30_Rh850Icus_AesDecrypt128.c, Cry_30_Rh850Icus_AesEncrypt128.c and so on.

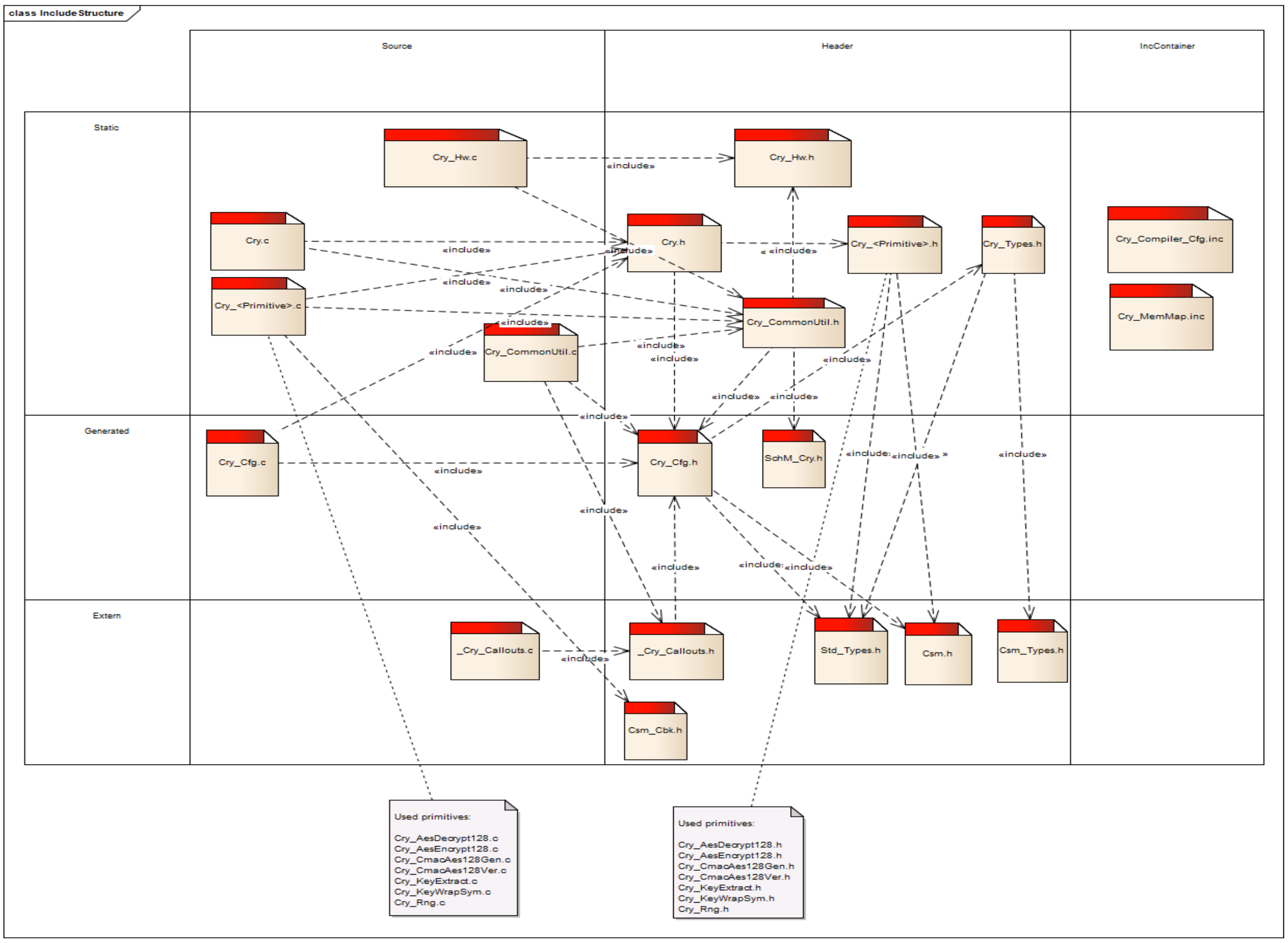

Figure 4-1 Include structure

4.4 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table (Table 4-4) contains the memory section names and the compiler abstraction definitions of the CRY_30_RH850ICUS and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	CRY_30_RH850ICUS_CODE	CRY_30_RH850ICUS_VAR_NOINIT	CRY_30_RH850ICUS_VAR_ZERO_INIT	CRY_30_RH850ICUS_APPL_VAR
CRY_30_RH850ICUS_START_SEC_CODE CRY_30_RH850ICUS_STOP_SEC_CODE	■			■
CRY_30_RH850ICUS_START_SEC_VAR_NOINIT_8BIT CRY_30_RH850ICUS_STOP_SEC_VAR_NOINIT_8BIT		■		
CRY_30_RH850ICUS_START_SEC_VAR_NOINIT_32BIT CRY_30_RH850ICUS_STOP_SEC_VAR_NOINIT_32BIT		■		
CRY_30_RH850ICUS_START_SEC_VAR_NOINIT_UNSPECIFIED CRY_30_RH850ICUS_STOP_SEC_VAR_NOINIT_UNSPECIFIED		■		
CRY_30_RH850ICUS_START_SEC_VAR_INIT_UNSPECIFIED CRY_30_RH850ICUS_STOP_SEC_VAR_INIT_UNSPECIFIED			■	

Table 4-4 Compiler abstraction and memory mapping

5 API Description

5.1 Interfaces Overview

For an interfaces overview please see Figure 2-2 'Figure 2-2: Interfaces to adjacent modules of the CRY_30_RH850ICUS'.

5.2 Type Definitions

The types defined by the CRY_30_RH850ICUS are described in this chapter.

Type Name	C-Type	Description	Value Range
Cry_SheTimeoutApiSectionType	enum	Values used for the parameter service of the location callout	`CRY_SHE_TO_SECTION_START_SERVICE` Start function of a service
			`CRY_SHE_TO_SECTION_UPDATE_SERVICE` Update function of a service
			`CRY_SHE_TO_SECTION_START_LOOP` Before entering a while loop which waits for a response from the SHE
			`CRY_SHE_TO_SECTION_STOP_LOOP` After leaving a while loop which waited for a response from the SHE
			`CRY_SHE_TO_SECTION_FINISH_SERVICE` Finish function of a service
			`CRY_SHE_TO_SECTION_SINGLE_CALL_SERVICE` Function of a service which consists of only one function (e.g. Cry_30_Rh850Icus_RngGenerate).
			`CRY_SHE_TO_SECTION_INIT_SERVICE` Init function of a service (e.g. Cry_30_Rh850Icus_RngInit)
Cry_SheTimeoutApiServiceType	enum	Values used for the parameter section of the location callout	`CRY_SHE_TO_SERVICE_CMAC_VERIFY`
			`CRY_SHE_TO_SERVICE_CMAC_GENERATE`
			`CRY_SHE_TO_SERVICE_AES_DECRYPT`
			`CRY_SHE_TO_SERVICE_AES_ENCRYPT`
			`CRY_SHE_TO_SERVICE_KEY_EXTRACT`
			`CRY_SHE_TO_SERVICE_KEY_WRAP`
			`CRY_SHE_TO_SERVICE_PRNG_SEED`
			`CRY_SHE_TO_SERVICE_PRNG_GENERATE`
			`CRY_SHE_TO_SERVICE_CANCEL`
			`CRY_SHE_TO_SERVICE_UNDEFINED`

Table 5-1 Type definitions

5.3 Structures

5.3.1 Configuration structures

Note
These structures and their content are generated and filled automatically by the configurator. It is not recommended to fill the structure manually.

5.3.1.1 Cry_30_Rh850Icus_AesEncrypt128ConfigType

This structure represents the configuration for the Cry_30_Rh850Icus_AesEncrypt128 service.

Struct Element Name	C-Type	Description	Value Range
BlockModeOfAesEncrypt128Config	uint32	Block mode.	`CRY_BLOCKMODE_ECB`, `CRY_BLOCKMODE_CBC`
KeyIdTypeOfAesEncrypt128Config	uint32	Defines the interpretation of the keyId	`CRY_KEYIDTYPE_MAPPED`, `CRY_KEYIDTYPE_RAW`
AesEncrypt128WorkSpaceIdxOfAesDecrypt128Config	uint8	Index of the workspace. The corresponding workspace is used as data storage for the primitives.	

Table 5-2 Cry_30_Rh850Icus_ AesEncrypt128ConfigType

5.3.1.2 Cry_30_Rh850Icus_AesDecrypt128ConfigType

This structure represents the configuration for the Cry_30_Rh850Icus_AesDecrypt128 service.

Struct Element Name	C-Type	Description	Value Range
BlockModeOfAesDecrypt128Config	uint32	Block mode.	`CRY_BLOCKMODE_ECB`, `CRY_BLOCKMODE_CBC`
KeyIdTypeOfAesDecrypt128Config	uint32	Defines the interpretation of the keyId.	`CRY_KEYIDTYPE_MAPPED`, `CRY_KEYIDTYPE_RAW`
AesDecrypt128WorkSpaceIdxOfAesDecrypt128Config	uint8	Index of the workspace. The corresponding workspace is used as data storage for the primitives.	

Table 5-3 Cry_30_Rh850Icus_AesDecrypt128ConfigType

5.3.1.3 Cry_30_Rh850Icus_CmacAes128GenConfigType

This structure represents the configuration for the Cry_30_Rh850Icus_CmacAes128Gen service.

Struct Element Name	C-Type	Description	Value Range
KeyIdTypeOfCmacAes128GenConfig	uint32	Defines the interpretation of the keyId.	`CRY_KEYIDTYPE_MAPPED`, `CRY_KEYIDTYPE_RAW`
CmacAes128GenWorkSpaceIdxOfCmacAes128GenConfig	uint8	Index of the workspace. The corresponding workspace is used as data storage for the primitives.	

Table 5-4 Cry_30_Rh850Icus_CmacAes128GenConfigType

5.3.1.4 Cry_30_Rh850Icus_CmacAes128VerConfigType

This structure represents the configuration for the Cry_30_Rh850Icus_CmacAes128Ver service.

Struct Element Name	C-Type	Description	Value Range
KeyIdTypeOfCmacAes128VerConfig	uint32	Defines the interpretation of the keyId.	`CRY_KEYIDTYPE_MAPPED`, `CRY_KEYIDTYPE_RAW`
lengthInBytes	boolean	Defines if Mac Length is interpreted in bytes or bits.	`CRY_MAC_LENGTH_IN_BYTES`, `CRY_MAC_LENGTH_IN_BITS`
CmacAes128VerWorkSpaceIdxOfCmacAes128GenConfig	uint8	Index of the workspace. The corresponding workspace is used as data storage for the primitives.	

Table 5-5 Cry_30_Rh850Icus_CmacAes128VerConfigType

5.3.1.5 Cry_30_Rh850Icus_KeyExtractConfigType

This structure represents the configuration for the Cry_30_Rh850Icus_KeyExtract service.

Struct Element Name	C-Type	Description	Value Range
KeyIdTypeOfKeyExtractConfig	uint32	Defines the interpretation of the keyId	`CRY_KEYIDTYPE_MAPPED`, `CRY_KEYIDTYPE_RAW`
KeyExtractWorkSpaceIdxOfKeyExtractConfig	uint8	Index of the workspace. The corresponding workspace is used as data storage for the primitives.	

Table 5-6 Cry_30_Rh850Icus_KeyExtractConfigType

5.3.1.6 Cry_30_Rh850Icus_KeyWrapSymConfigType

This structure represents the configuration for the Cry_30_Rh850Icus_KeyWrapSym service.

Struct Element Name	C-Type	Description	Value Range
KeyIdTypeOfKeyWrapSymConfig	uint32	Defines the interpretation of the keyId	`CRY_KEYIDTYPE_MAPPED`, `CRY_KEYIDTYPE_RAW`
KeyWrapSymWorkSpaceIdxOfKeyWrapSymConfig	uint8	Index of the workspace. The corresponding workspace is used as data storage for the primitives.	

Table 5-7 Cry_30_Rh850Icus_KeyWrapSymConfigType

5.3.1.7 Cry_30_Rh850Icus_RngConfigType

This structure represents the configuration for the Cry_30_Rh850Icus_Rng service.

Struct Element Name	C-Type	Description	Value Range
RngWorkSpaceIdxOfRngConfig	uint8	Index of the workspace. The corresponding workspace is used as data storage for the primitives.	

Table 5-8 Cry_30_Rh850Icus_RngConfigType

5.4 Services provided by CRY_30_RH850ICUS

5.4.1 Cry_30_Rh850Icus_Init

Prototype	
`void **Cry_30_Rh850Icus_Init** (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Initializes the Cry. Initialize the enabled services and set the global service state to idle.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > This function has to be called during start-up.	
Call Context	
> This function can be called from interrupt level or from task level.	

Table 5-9 Cry_30_Rh850Icus_Init

5.4.2 Cry_30_Rh850Icus_InitMemory

Prototype	
`void **Cry_30_Rh850Icus_InitMemory** (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Service to initialize module global variables at power up. This function initializes the variables in *_INIT_* sections. Used in case they are not initialized by the startup code.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	
Call Context	
> This function can be called from task level only.	

Table 5-10 Cry_30_Rh850Icus_InitMemory

5.4.3 Cry_30_Rh850Icus_GetVersionInfo

Prototype	
`void` **`Cry_30_Rh850Icus_GetVersionInfo`** `(Std_VersionInfoType *cryVerInfoPtr)`	
Parameter	
`cryVerInfoPtr`	Pointer where to store the version information of this module. Parameter must not be NULL.
Return code	
void	none
Functional Description	
This function retrieves the version information of the Cry module. It stores the version information, i.e. Module Id, Vendor Id, Vendor specific version numbers to structure pointed by cryVerInfoPtr.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The availability of this service depends on CRY_30_RH850ICUS_VERSION_INFO_API.	
Call Context	
> This function can be called from task level only.	

Table 5-11 Cry_30_Rh850Icus_GetVersionInfo

5.4.4 Cry_30_Rh850Icus_AesEncrypt128Start

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_AesEncrypt128Start`** `(const void *cfgPtr, const Csm_SymKeyType *keyPtr, const uint8 *InitVectorPtr, uint32 InitVectorLength)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_AesEncrypt128ConfigType for more information.
keyPtr	Holds a pointer to the key which has to be used during the symmetrical encryption. The keyPtr can either contain a plaintext key (keyPtr.length = 16) or a keyId (keyPtr.length = 1).
InitVectorPtr	Holds a pointer to initialization vector which has to be used during the symmetrical encryption.
InitVectorLength	Holds the length of the initialization vector in bytes. Only the value 16 (for CBC mode) and 0 (for ECB) is allowed.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the symmetrical encryption service of the module. **Synchronous configuration:** The function starts the service. The keyId is stored and if the keyPtr contains a plaintext, the key is loaded to the RAM key slot. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
Caution If this function is called successful (CSM_E_OK), the caller must call Cry_30_Rh850Icus_AesEncrypt128Finish hereafter. A call of Cry_30_Rh850Icus_AesEncrypt128Update between them is allowed. Attention: If the Finish-function is never called, the SHE is blocked for all other services. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_AESENCRYPT128CONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-12 Cry_30_Rh850Icus_AesEncrypt128Start

5.4.5 Cry_30_Rh850Icus_AesEncrypt128Update

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_AesEncrypt128Update`** `(Const void *cfgPtr, const uint8 *plainTextPtr, uint32 plainTextLength, uint8 *cipherTextPtr, uint32 *cipherTextLengthPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_AesEncrypt128ConfigType for more information.
plainTextPtr	Holds a pointer to the data for which an encrypted text shall be computed. The address of the pointer needs to be aligned on 32-bit.
plainTextLength	Contains the number of bytes for which the encrypted text shall be computed. Only values which are the same or a multiple of the block size (16 bytes) are allowed.
cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text. The address of the pointer needs to be aligned on 32-bit.
cipherTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. The buffer size must have at least the size of the plainTextLength. When the request has finished successful, the length of the returned encrypted is stored.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_SMALL_BUFFER	The provided buffer is too small to store the result.
Functional Description	
This interface passes input data to the symmetrical encryption service. **Synchronous configuration:** The function updates the service. The command to encrypt the plaintext is send to the SHE. If no error occurred, the cipherTextLengthPtr gets updated with the length of the encrypted text. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be updated in the next main-function call.	
Particularities and Limitations	

Note It’s not possible to call Cry_30_Rh850Icus_AesEncrypt128Update multiple times in order to feed the symmetrical encryption service with separate input data chunks. Therefore plainTextLength must be set to the length of the complete input data. The data has to be padded to the length of the block size (16 bytes) or a multiple of it before calling this function. **Caution** The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_AesEncrypt128Start with return value CSM_E_OK). > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_AESENCRYPT128CONFIG.
Call Context
> This function can be called from task level only.

Table 5-13 Cry_30_Rh850Icus_AesEncrypt128Update

5.4.6 Cry_30_Rh850Icus_AesEncrypt128Finish

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_AesEncrypt128Finish`** `(Const void *cfgPtr, uint8 *cipherTextPtr, uint32 *cipherTextLengthPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_AesEncrypt128ConfigType for more information.
cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
cipherTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. When the request has finished successful, the length of the returned encrypted text is stored.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_SMALL_BUFFER	The provided buffer is too small to store the result.
Functional Description	
This interface shall be used to finish the symmetrical encryption service. **Synchronous configuration:** The function finishes the service. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be finished in the next main-function call.	
Particularities and Limitations	
Caution The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_AesEncrypt128Start with return value CSM_E_OK). A function call of Cry_30_Rh850Icus_AesEncrypt128Update between them is allowed. This function must only be called once after a successful call of Cry_30_Rh850Icus_AesEncrypt128Start. It is not allowed to call this function multiple times, even if the return value is not CSM_E_OK. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_AESENCRYPT128CONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-14 Cry_30_Rh850Icus_AesEncrypt128Finish

5.4.7 Cry_30_Rh850Icus_AesEncrypt128MainFunction

Prototype	
`void` **`Cry_30_Rh850Icus_AesEncrypt128MainFunction`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_AESENCRYPT128CONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-15 Cry_30_Rh850Icus_AesEncrypt128MainFunction

5.4.8 Cry_30_Rh850Icus_AesDecrypt128Start

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_AesDecrypt128Start`** `(Const void *cfgPtr, const Csm_SymKeyType *keyPtr, const uint8 *InitVectorPtr, uint32 InitVectorLength)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_AesDecrypt128ConfigType for more information.
keyPtr	Holds a pointer to the key which has to be used during the symmetrical decryption. The keyPtr can either contain a plaintext key (keyPtr.length = 16) or a keyId (keyPtr.length = 1).
InitVectorPtr	Holds a pointer to initialization vector which has to be used during the symmetrical decryption.
InitVectorLength	Holds the length of the initialization vector in bytes. Only the value 16 (for CBC mode) and 0 (for ECB) is allowed.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the symmetrical decryption service of the module. **Synchronous configuration:** The function starts the service. The keyId is stored and if the keyPtr contains a plaintext, the key is loaded to the RAM key slot. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
Caution If this function is called successful (CSM_E_OK), the caller must call Cry_30_Rh850Icus_AesEncrypt128Finish hereafter. A call of Cry_30_Rh850Icus_AesEncrypt128Update between them is allowed. Attention: If the Finish-function is never called, the SHE is blocked for all other services. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_AESDECRYPT128CONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-16 Cry_30_Rh850Icus_AesDecrypt128Start

5.4.9 Cry_30_Rh850Icus_AesDecrypt128Update

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_AesDecrypt128Update`** `(Const void *cfgPtr, const uint8 *cipherTextPtr, uint32 cipherTextLength, uint8 *plainTextPtr, uint32 *plainTextLengthPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_AesDecrypt128ConfigType for more information.
cipherTextPtr	Holds a pointer to the data for which a decrypted text shall be computed. The address of the pointer needs to be aligned on 32-bit.
cipherTextLength	Contains the number of bytes for which the decrypted text shall be computed. Only values which are the same or a multiple of the block size (16 bytes) are allowed.
plainTextPtr	Holds a pointer to the memory location which will hold the encrypted text. The address of the pointer needs to be aligned on 32-bit.
plainTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. The buffer size must have at least the size of the cipherTextLength. When the request has finished successful, the length of the returned decrypted is stored.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_SMALL_BUFFER	The provided buffer is too small to store the result.
Functional Description	
This interface passes input data to the symmetrical decryption service. **Synchronous configuration:** The function updates the service. The command to decrypt the ciphertext is send to the SHE. If no error occurred, the plainTextLengthPtr gets updated with the length of the decrypted text. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be updated in the next main-function call.	
Particularities and Limitations	

Note
It’s not possible to call Cry_30_Rh850Icus_AesDecrypt128Update multiple times in order to feed the symmetrical encryption service with separate input data chunks.

Therefore cipherTextLength must be set to the length of the complete input data.

The data has to be padded to the length of the block size (16 bytes) or a multiple of it before calling this function.

Caution
The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_AesDecrypt128Start with return value CSM_E_OK).

> This function can be synchronous or asynchronous.
> This function is non-reentrant.
> This function is called by the CSM.
> The availability of this service depends on CRY_30_RH850ICUS_AESENCRYPT128CONFIG.

Call Context
> This function can be called from task level only.

Table 5-17 Cry_30_Rh850Icus_AesDecrypt128Update

5.4.10 Cry_30_Rh850Icus_AesDecrypt128Finish

Prototype	
`Csm_ReturnType **Cry_30_Rh850Icus_AesDecrypt128Finish** (Const void *cfgPtr, uint8 *plainTextPtr, uint32 *plainTextLengthPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_AesDecrypt128ConfigType for more information.
plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
plainTextLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the provided buffer. When the request has finished successful, the length of the returned decrypted text is stored.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_SMALL_BUFFER	The provided buffer is too small to store the result.
Functional Description	
This interface shall be used to finish the symmetrical decryption service. **Synchronous configuration:** The function finishes the service. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be finished in the next main-function call.	
Particularities and Limitations	
Caution The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_AesDecrypt128Start with return value CSM_E_OK). A function call of call Cry_30_Rh850Icus_AesDecrypt128Update between them is allowed. This function must only be called once after a successful call of Cry_30_Rh850Icus_AesDecrypt128Start. It is not allowed to call this function multiple times, even if the return value is not CSM_E_OK. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_AESDECRYPT128CONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-18 Cry_30_Rh850Icus_AesDecrypt128Finish

5.4.11 Cry_30_Rh850Icus_AesDecrypt128MainFunction

Prototype	
`void` **`Cry_30_Rh850Icus_AesDecrypt128MainFunction`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_AESDECRYPT128CONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-19 Cry_30_Rh850Icus_AesDecrypt128MainFunction

5.4.12 Cry_30_Rh850Icus_CmacAes128GenStart

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_CmacAes128GenStart`** `(Const void *cfgPtr, const Csm_SymKeyType *keyPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_CmacAes128GenConfigType for more information.
keyPtr	Holds a pointer to the key which has to be used for the CMAC generation. The keyPtr can either contain a plaintext key (keyPtr.length = 16) or a keyId (keyPtr.length = 1).
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the CMAC generation service of this module. **Synchronous configuration:** The function starts the service. The keyId is stored and if the keyPtr contains a plaintext, the key is loaded to the RAM key slot. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
Caution If this function is called successful (CSM_E_OK), the caller must call Cry_30_Rh850Icus_CmacAes128GenFinish hereafter. A call of Cry_30_Rh850Icus_CmacAes128GenUpdate between them is allowed. Attention: If the Finish-function is never called, the SHE is blocked for all other services. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_CMACAES128GENCONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-20 Cry_30_Rh850Icus_CmacAes128GenStart

5.4.13 Cry_30_Rh850Icus_CmacAes128GenUpdate

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_CmacAes128GenUpdate`** `(Const void *cfgPtr, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_CmacAes128GenConfigType for more information.
dataPtr	Holds a pointer to the data for which a CMAC shall be computed. The address of the pointer needs to be aligned on 32-bit.
dataLength	Contains the number of bytes for which the MAC shall be computed.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
Functional Description	
This interface passes input data for the CMAC generation. **Synchronous configuration:** The function updates the service. The command to generate the CMAC is send to the SHE. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be updated in the next main-function call.	
Particularities and Limitations	
Note Due to limitations in the API of the SHE, it's not possible to call Cry_30_Rh850Icus_CmacAes128GenUpdate multiple times in order to feed the CMAC generation with separate input data chunks. Therefore dataLength must be set to the length of the complete input data. **Caution** The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_CmacAes128GenStart with return value CSM_E_OK). > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_CMACAES128GENCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-21 Cry_30_Rh850Icus_CmacAes128GenUpdate

5.4.14 Cry_30_Rh850Icus_CmacAes128GenFinish

Prototype	
`Csm_ReturnType **Cry_30_Rh850Icus_CmacAes128GenFinish** (Const void *cfgPtr, const uint8 *resultPtr, uint32* resultLengthPtr, boolean truncationIsAllowed)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_CmacAes128GenConfigType for more information.
resultPtr	Holds a pointer to the memory location which will hold the result of the CMAC generation. If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated
resultLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by resultPtr. When the request has finished successful, the length of the returned MAC shall be stored.
truncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: truncation is allowed. FALSE: truncation is not allowed.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed
CSM_E_SMALL_BUFFER	The provided buffer is too small to store the result, and truncation was not allowed.
Functional Description	
This interface shall be used to finish the SHE-CMAC generation service. **Synchronous configuration:** The function finishes the service. If no error occurred, the CMAC is written to the resultPtr and the resultLengthPtr gets updated with the length of the CMAC. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be finished in the next main-function call.	
Particularities and Limitations	

Caution The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_CmacAes128GenStart with return value CSM_E_OK). A function call of call Cry_30_Rh850Icus_CmacAes128GenUpdate between them is allowed. This function must only be called once after a successful call of Cry_30_Rh850Icus_CmacAes128GenStart. It is not allowed to call this function multiple times, even if the return value is not CSM_E_OK. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_CMACAES128GENCONFIG.
Call Context
> This function can be called from task level only.

Table 5-22 Cry_30_Rh850Icus_CmacAes128GenFinish

5.4.15 Cry_30_Rh850Icus_CmacAes128GenMainFunction

Prototype	
`void **Cry_30_Rh850Icus_CmacAes128GenMainFunction** (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_CMACAES128GENCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-23 Cry_30_Rh850Icus_CmacAes128GenMainFunction

5.4.16 Cry_30_Rh850Icus_CmacAes128VerStart

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_CmacAes128VerStart`** `(Const void *cfgPtr, const Csm_SymKeyType *keyPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_CmacAes128VerConfigType for more information.
keyPtr	Holds a pointer to the key which has to be used for the CMAC verification. The keyPtr can either contain a plaintext key (keyPtr.length = 16) or a keyId (keyPtr.length = 1).
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the CMAC verification service of this module. **Synchronous configuration:** The function starts the service. The keyId is stored and if the keyPtr contains a plaintext, the key is loaded to the RAM key slot. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
Caution If this function is called successful (CSM_E_OK), the caller must call Cry_30_Rh850Icus_CmacAes128VerFinish hereafter. A call of Cry_30_Rh850Icus_CmacAes128VerUpdate between them is allowed. Attention: If the Finish-function is never called, the SHE is blocked for all other services. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_CMACAES128VERCONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-24 Cry_30_Rh850Icus_CmacAes128VerStart

5.4.17 Cry_30_Rh850Icus_CmacAes128VerUpdate

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_CmacAes128VerUpdate`** `(Const void *cfgPtr, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_CmacAes128VerConfigType for more information.
dataPtr	Holds a pointer to the data for which a CMAC shall be verified. The address of the pointer needs to be aligned on 32-bit. See important note for this parameter below.
dataLength	Contains the number of bytes for which the CMAC shall be verified.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.

Functional Description

This interface passes the input data to the CMAC verification service.

Synchronous configuration:

The function updates the service. The dataLength and dataPtr are stored internal for the finish call.

Asynchronous configuration:

The parameters are stored in a buffer, and the service gets marked to be updated in the next main-function call.

Particularities and Limitations

Note
Due to limitations in the API of the SHE, it's not possible to call Cry_30_Rh850Icus_CmacAes128VerUpdate multiple times in order to feed the CMAC verification with separate input data chunks.

Therefore dataLength must be set to the length of the complete input data.

Caution
The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_CmacAes128VerStart with return value CSM_E_OK).

The function stores only the address of the dataPtr and not the content the pointer references. The content of the dataPtr is valid until Cry_30_Rh850Icus_CmacAes128VerFinish has called and executed.

> This function can be synchronous or asynchronous.
> This function is non-reentrant.
> This function is called by the CSM.
> The availability of this service depends on CRY_30_RH850ICUS_CMACAES128VERCONFIG.

Call Context

> This function can be called from task level only.

Table 5-25 Cry_30_Rh850Icus_CmacAes128VerUpdate

5.4.18 Cry_30_Rh850Icus_CmacAes128VerFinish

Prototype

```
Csm_ReturnType Cry_30_Rh850Icus_CmacAes128VerFinish (Const void
*cfgPtr, const uint8 *MacPtr, uint32 MacLength, Csm_VerifyResultType
*resultPtr)
```

Parameter

cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_CmacAes128VerConfigType for more information.
MacPtr	Holds a pointer to the memory location which will hold the CMAC to verify. The address of the pointer needs to be aligned on 32-bit.
MacLength	Holds the length of the MAC to be verified. Depending on the configuration, this value is interpreted as number of bits or number of bytes to verify. The maximum supported length is 16 Byte, respectively 128 Bit.
resultPtr	Holds a pointer to the memory location which will hold the CMAC.

Return code

CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed

Functional Description

This interface shall be used to finish the CMAC verification.

Synchronous configuration:

The function finishes the service.

The command to verify the CMAC is send to the SHE. The result of the verification is written to the resultPtr.

Asynchronous configuration:

The parameters are stored in a buffer, and the service gets marked to be finished in the next main-function call.

Particularities and Limitations

Caution
The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_CmacAes128VerStart with return value CSM_E_OK). A function call of Cry_30_Rh850Icus_CmacAes128VerUpdate between them is allowed.

This function must only be called once after a successful call of Cry_30_Rh850Icus_CmacAes128VerStart. It is not allowed to call this function multiple times, even if the return value is not CSM_E_OK.

The caller of this function needs to ensure, that the dataPtr and its data which was passed in the update function is still valid.

> This function can be synchronous or asynchronous.
> This function is non-reentrant.
> This function is called by the CSM.
> The availability of this service depends on CRY_30_RH850ICUS_CMACAES128VERCONFIG.

Call Context
> This function can be called from task level only.

Table 5-26 Cry_30_Rh850Icus_CmacAes128VerFinish

5.4.19 Cry_30_Rh850Icus_CmacAes128VerMainFunction

Prototype	
`void` **`Cry_30_Rh850Icus_CmacAes128VerMainFunction`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_CMACAES128GENCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-27 Cry_30_Rh850Icus_CmacAes128VerMainFunction

5.4.20 Cry_30_Rh850Icus_KeyExtractStart

Prototype	
`void Cry_30_Rh850Icus_KeyExtractStart (Csm_ConfigIdType cfgId)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_KeyExtractConfigType for more information.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the KeyExtract service of the module. **Synchronous configuration:** The function starts the service. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
Caution If this function is called successful (CSM_E_OK), the caller must call Cry_30_Rh850Icus_KeyExtractFinish hereafter. A call of Cry_30_Rh850Icus_KeyExtractUpdate between them is allowed. Attention: If the Finish-function is never called, the SHE is blocked for all other services. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_KEYEXTRACTCONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-28 Cry_30_Rh850Icus_KeyExtractStart

5.4.21 Cry_30_Rh850Icus_KeyExtractUpdate

Prototype	
`void` **`Cry_30_Rh850Icus_KeyExtractUpdate`** `(Csm_ConfigIdType cfgId, const uint8* dataPtr, uint32 dataLength)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_KeyExtractConfigType for more information.
dataPtr	Holds a pointer to the data which contains either - a plaintext key (Length is 16) - Messages M1, M2, and M3 with an optional prepending KeyId to update a keyslot in the SHE. (Length is 64 or 65 Bytes) - a KeyId (Length is 1)
dataLength	Holds the length of the data in bytes.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.

Functional Description

This interface passes input data to the key extraction service.

Synchronous configuration:

The function updates the service. This function provides different possibilities to store keys.

1. Input data with the length of 16 Byte are stored in the KEY_RAM.
2. Input data with the length of 64 Byte are interpreted as M1-M3 for the key update protocol as specified in the SHE spec.
3. Input data with the length of 65 Byte are interpreted as M1-M3 for the key update protocol as specified in the SHE spec including a KeyId in the first byte of the input data.
4. Input data with the length of 1 Byte are used as the data in key which is returned in the Finish-Function.

Asynchronous configuration:

The parameters are stored in a buffer, and the service gets marked to be updated in the next main-function call.

Particularities and Limitations

Note
The driver does not support multiple valid calls of Cry_30_Rh850Icus_KeyExtractUpdate.

Caution
The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_KeyExtractStart with return value CSM_E_OK).

- This function can be synchronous or asynchronous.
- This function is non-reentrant.
- This function is called by the CSM.
- The availability of this service depends on CRY_30_RH850ICUS_KEYEXTRACTCONFIG.

Call Context
> This function can be called from task level only.

Table 5-29 Cry_30_Rh850Icus_KeyExtractUpdate

5.4.22 Cry_30_Rh850Icus_KeyExtractFinish

Prototype	
`void` **`Cry_30_Rh850Icus_KeyExtractFinish`** `(Csm_ConfigIdType cfgId, Csm_SymKeyType* keyPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_KeyExtractConfigType for more information.
keyPtr	Holds a pointer to a structure where the result (e.g. the symmetrical key) is stored in.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
Functional Description	
This interface shall be used to finish the key extract service. **Synchronous configuration:** The function finishes the service. If a key is updated in the NVM, they key.data will contain the M4-M5 of the SHE key update protocol. If the KEY_RAM has been updated, the key.data will contain the corresponding keyId for that slot which can be used as input for e.g. AES encrypt. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be finished in the next main-function call.	
Particularities and Limitations	
Caution The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_KeyExtractStart with return value CSM_E_OK). A function call of call Cry_30_Rh850Icus_KeyExtractUpdate between them is allowed. This function must only be called once after a successful call of Cry_30_Rh850Icus_KeyExtractStart. It is not allowed to call this function multiple times, even if the return value is not CSM_E_OK. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_KEYEXTRACTCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-30 Cry_30_Rh850Icus_KeyExtractFinish

5.4.23 Cry_30_Rh850Icus_KeyExtractMainFunction

Prototype	
`void` **`Cry_30_Rh850Icus_KeyExtractMainFunction`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_KEYEXTRACTCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-31 Cry_30_Rh850Icus_KeyExtractMainFunction

5.4.24 Cry_30_Rh850Icus_KeyWrapSymStart

Prototype	
`void Cry_30_Rh850Icus_KeyWrapSymStart (Csm_ConfigIdType cfgId, const Csm_SymKeyType * keyPtr, const Csm_SymKeyType * wrappingKeyPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_KeyWrapSymConfigType for more information.
`keyPtr`	Holds a pointer to the symmetric key to be wrapped.
`wrappingKeyPtr`	Holds a pointer to the key used for wrapping.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the symmetrical key wrapping service of the module. **Synchronous configuration:** The function starts the service. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
Caution If this function is called successful (CSM_E_OK), the caller must call Cry_30_Rh850Icus_KeyWrapSymFinish hereafter. A call of Cry_30_Rh850Icus_KeyWrapSymUpdate between them is allowed. Attention: If the Finish-function is never called, the SHE is blocked for all other services. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_KEYWRAPSYMCONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-32 Cry_30_Rh850Icus_KeyWrapSymStart

5.4.25 Cry_30_Rh850Icus_KeyWrapSymUpdate

Prototype	
`void Cry_30_Rh850Icus_KeyWrapSymUpdate (Csm_ConfigIdType cfgId, const uint8* dataPtr, uint32 * dataLengthPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_KeyWrapSymConfigType for more information.
dataPtr	Holds a pointer to the memory location which will hold the result of the key wrapping. The address of the pointer needs to be aligned on 32-bit.
dataLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by dataPtr. When the request has finished successful, the length of the computed value is be stored. The only supported valid value for this parameter is 112.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
Functional Description	
This interface shall be used to retrieve the result of the key wrapping operation. **Synchronous configuration:** The function updates the service. The command to export the RAM key is send to the SHE. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be updated in the next main-function call.	
Particularities and Limitations	
Note The driver does not support multiple valid calls of Cry_30_Rh850Icus_KeyWrapSymUpdate. **Caution** The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_KeyWrapSymStart with return value CSM_E_OK). > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_KEYWRAPSYMCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-33 Cry_30_Rh850Icus_KeyWrapSymUpdate

5.4.26 Cry_30_Rh850Icus_KeyWrapSymFinish

Prototype	
`void` **`Cry_30_Rh850Icus_KeyWrapSymFinish`** `(Csm_ConfigIdType cfgId)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_KeyWrapSymConfigType for more information.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
Functional Description	
This interface shall be used to finish the symmetrical key wrapping service. **Synchronous configuration:** The function finishes the service. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be finished in the next main-function call.	
Particularities and Limitations	
Caution The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_KeyWrapSymStart with return value CSM_E_OK). A function call of call Cry_30_Rh850Icus_KeyWrapSymUpdate between them is allowed. This function must only be called once after a successful call of Cry_30_Rh850Icus_KeyWrapSymStart. It is not allowed to call this function multiple times, even if the return value is not CSM_E_OK. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_KEYWRAPSYMCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-34 Cry_30_Rh850Icus_KeyWrapSymFinish

5.4.27 Cry_30_Rh850Icus_KeyWrapSymMainFunction

Prototype	
`void` **`Cry_30_Rh850Icus_KeyWrapSymMainFunction`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_KEYWRAPSYMCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-35 Cry_30_Rh850Icus_KeyWrapSymMainFunction

5.4.28 Cry_30_Rh850Icus_RngSeedStart

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_RngSeedStart`** `(Const void *cfgPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_RngConfigType for more information.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the random number generator seed service of the module. **Synchronous configuration:** The function starts the service. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
! **Caution** If this function is called successful (CSM_E_OK), the caller must call Cry_30_Rh850Icus_RngSeedFinish hereafter. A call of Cry_30_Rh850Icus_RngSeedUpdate between them is allowed. Attention: If the Finish-function is never called, the SHE is blocked for all other services. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_RNGCONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-36 Cry_30_Rh850Icus_RngSeedStart

5.4.29 Cry_30_Rh850Icus_RngSeedUpdate

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_RngSeedUpdate`** `(Const void *cfgPtr, const uint8 *seedPtr, uint32 seedLength)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_RngConfigType for more information.
seedPtr	Holds a pointer to the seed for the random number generator. The address of the pointer needs to be aligned on 32-bit.
seedLength	Contains the length of the seed in bytes. Only the value 16 is supported.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
Functional Description	
This interface passes a seed to the random number generator seed service. **Synchronous configuration:** The function updates the service. The command to extend the seed with the given seedPtr is send to the hardware. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be updated in the next main-function call.	
Particularities and Limitations	
Note The driver does not support multiple valid calls of Cry_30_Rh850Icus_RngSeedUpdate. **Caution** The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_RngSeedStart with return value CSM_E_OK). > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_RNGCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-37 Cry_30_Rh850Icus_RngSeedUpdate

5.4.30 Cry_30_Rh850Icus_RngSeedFinish

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_RngSeedFinish`** `(Const void *cfgPtr)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_RngConfigType for more information.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
Functional Description	
This interface shall be used to finish the random number generator seed service. **Synchronous configuration:** The function finishes the service. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be finished in the next main-function call.	
Particularities and Limitations	
Caution The function must only be called, if the service was started before successful (call of Cry_30_Rh850Icus_RngSeedStart with return value CSM_E_OK). A function call of call Cry_30_Rh850Icus_RngSeedUpdate between them is allowed. This function must only be called once after a successful call of Cry_30_Rh850Icus_KeyWrapSymStart. It is not allowed to call this function multiple times, even if the return value is not CSM_E_OK. > This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_RNGCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-38 Cry_30_Rh850Icus_RngSeedFinish

5.4.31 Cry_30_Rh850Icus_RngSeedMainFunction

Prototype	
`void` Cry_30_Rh850Icus_**`RngSeedMainFunction`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_RNGCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-39 Cry_30_Rh850Icus_RngSeedMainFunction

5.4.32 Cry_30_Rh850Icus_RngGenerate

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_RngGenerate`** `(Const void *cfgPtr, uint8 *resultPtr, uint32 resultLength)`	
Parameter	
cfgPtr	Holds a pointer to the configuration of this service. See Cry_30_Rh850Icus_RngConfigType for more information.
resultPtr	Holds a pointer to the memory location which will hold the result of the random number generation. The memory location must have at least the size "resultLength".
resultLength	Holds the amount of random bytes which should be generated. The maximum supported size is 16 Byte.
Return code	
CSM_E_OK	Request successful.
CSM_E_NOT_OK	Request failed.
CSM_E_BUSY	Request failed, service is still busy.
Functional Description	
This interface shall be used to initialize the random number generator according to the SHE specification. **Synchronous configuration:** The command to generate a random number is send to the SHE. The generated random number is written to the resultPtr. **Asynchronous configuration:** The parameters are stored in a buffer, and the service gets marked to be started in the next main-function call.	
Particularities and Limitations	
> This function can be synchronous or asynchronous. > This function is non-reentrant. > This function is called by the CSM. > The availability of this service depends on CRY_30_RH850ICUS_RNGCONFIG. > Precondition: Service is idle.	
Call Context	
> This function can be called from task level only.	

Table 5-40 Cry_30_Rh850Icus_RngGenerate

5.4.33 Cry_30_Rh850Icus_RngGenerateMainFunction

Prototype	
`void` **`Cry_30_Rh850Icus_RngGenerateMainFunction`** `(void)`	
Parameter	
-	
Return code	
-	
Functional Description	
Implements the API to be called cyclically to process the requested service. This function needs to be called for asynchronous function handling in order to process the service. If a service is processed, the function sends a callback notification. **Note** This function is empty if 'Use Sync Job Processing' is enabled.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > This function has to be called by CSM. > This function must not be called by the application. > The availability of this service depends on CRY_30_RH850ICUS_RNGCONFIG.	
Call Context	
> This function can be called from task level only.	

Table 5-41 Cry_30_Rh850Icus_RngGenerateMainFunction

5.4.34 Cry_30_Rh850Icus_SelfTest

Prototype	
`Csm_ReturnType` **`Cry_30_Rh850Icus_SelfTest`** `(void)`	
Parameter	
-	
Return code	
CSM_E_OK	Self test was successful.
CSM_E_BUSY	Self test has not been performed, because the hardware was busy.
CSM_E_NOT_OK	Self test has failed
Functional Description	
Implements the API to perform a self check of the CMAC verifyication function. This is done by checking predefined test vectors. The self check is done once in the init function of the CRY. The user can perfom more self checks cyclically once the CRY is initialized.	
Particularities and Limitations	
> This function is synchronous. > This function is not reentrant. > The availability of this service depends on CRY_30_RH850ICUS_SELF_TEST.	
Call Context	
> This function can be called from task level only.	

Table 5-42 Cry_30_Rh850Icus_SelfTest

5.5 Configurable Interfaces

5.5.1 Callout Functions

At its configurable interfaces the CRY defines callout functions. The declarations of the callout functions are provided by the BSW module, i.e. the CRY. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. Additionally, some callouts can be enabled with the configuration. An example implementation is provided in the template files. The CRY callout function declarations are described in the following tables:

5.5.1.1 Timeout-API Location Callout

Prototype	
`void` **`[Location Callout Function Name]`** `(Cry_SheTimeoutApiServiceType service, Cry_SheTimeoutApiSectionType section)`	
Parameter	

`service`	This parameter provides information from which service the callout is called.
	
	The following services can be the source of the callout:
	`CRY_SHE_TO_SERVICE_CMAC_VERIFY`
	`CRY_SHE_TO_SERVICE_CMAC_GENERATE`
	`CRY_SHE_TO_SERVICE_AES_DECRYPT`
	`CRY_SHE_TO_SERVICE_AES_ENCRYPT`
	`CRY_SHE_TO_SERVICE_KEY_EXTRACT`
	`CRY_SHE_TO_SERVICE_KEY_WRAP`
	`CRY_SHE_TO_SERVICE_PRNG_SEED`
	`CRY_SHE_TO_SERVICE_PRNG_GENERATE`
	`CRY_SHE_TO_SERVICE_CANCEL`
	`CRY_SHE_TO_SERVICE_UNDEFINED`

`section`	This parameter provides information where exactly in the service, the callout is called.
	
	The following services can be the source of the callout:
	`CRY_SHE_TO_SECTION_START_SERVICE`
	`CRY_SHE_TO_SECTION_UPDATE_SERVICE,`
	`CRY_SHE_TO_SECTION_START_LOOP,`
	`CRY_SHE_TO_SECTION_STOP_LOOP,`
	`CRY_SHE_TO_SECTION_FINISH_SERVICE,`
	`CRY_SHE_TO_SECTION_SINGLE_CALL_SERVICE`
	`CRY_SHE_TO_SECTION_INIT_SERVICE`

Functional Description	
The function provides information about the service which is currently processed and at which location the callout is called.	
Particularities and Limitations	
> Function is only called when the timeout API is enabled.	
Call context	
> This function is called from task level only.	

Table 5-43 [Timeout-API location callout]

5.5.1.2 Timeout-API Loop Callout

Prototype	
`void` **`[Loop Callout Function Name]`** `(Cry_SheTimeoutApiServiceType service)`	
Parameter	

`service`	This parameter provides information from which service the callout is called.
	
	The following services can be the source of the callout:
	`CRY_SHE_TO_SERVICE_CMAC_VERIFY`
	`CRY_SHE_TO_SERVICE_CMAC_GENERATE`
	`CRY_SHE_TO_SERVICE_AES_DECRYPT`
	`CRY_SHE_TO_SERVICE_AES_ENCRYPT`
	`CRY_SHE_TO_SERVICE_KEY_EXTRACT`
	`CRY_SHE_TO_SERVICE_KEY_WRAP`
	`CRY_SHE_TO_SERVICE_PRNG_SEED`
	`CRY_SHE_TO_SERVICE_PRNG_GENERATE`
	`CRY_SHE_TO_SERVICE_CANCEL`
	`CRY_SHE_TO_SERVICE_UNDEFINED`

Return code	
`Csm_ReturnType`	To indicate that no timeout has occurred, return `CSM_E_OK`. If a timeout has occurred, return `CSM_E_NOT_OK`.
Functional Description	
This function is called in all loops which wait for a response from the SHE. When a timeout occurs (return code equals `CSM_E_NOT_OK`), the loop will be left and the command `CMD_CANCEL` will be sent to the SHE to stop the currently executing command on the SHE.	
Particularities and Limitations	
> Function is only called when the timeout API is enabled.	
Call context	
> This function is called from task level only.	

Table 5-44 [Timeout-API Loop Callout]

5.5.1.3 Cry_30_Rh850Icus_HardwareErrorCode_Callout

Prototype	
`void` **`Cry_30_Rh850Icus_HardwareErrorCode_Callout`** `(void)`	
Parameter	
`error`	Error code of the Hardware
Return code	
–	

Functional Description
This function gets the error code of the hardware. This error code can for example be passed to the mode management.
Particularities and Limitations
> Function is only called when CRY_30_RH850ICUS_HARDWARE_ERROR_CODE is STD_ON.
Call context
> This function is called from task level only.

Table 5-45 Cry_30_Rh850Icus_HardwareErrorCode_Callout

5.5.1.4 Cry_30_Rh850Icus_DataFlashReadStart_Callout

Prototype	
`boolean` **`Cry_30_Rh850Icus_DataFlashReadStart_Callout`** `(void)`	
Parameter	
–	
Return code	
`boolean`	TRUE if a read access to the data flash is granted, otherwise false.
Functional Description	
This function shall try to get a read lock for the data flash.	
Particularities and Limitations	
> Function is only called when CRY_30_RH850ICUS_DATA_FLASH_SYNCHRONISATION is STD_ON.	
Call context	
> This function is called from task level only.	

Table 5-46 Cry_30_Rh850Icus_DataFlashReadStart_Callout

5.5.1.5 Cry_30_Rh850Icus_DataFlashReadEnd_Callout

Prototype	
`void` **`Cry_30_Rh850Icus_DataFlashReadEnd_Callout`** `(void)`	
Parameter	
–	
Return code	
–	
Functional Description	
This function shall try to release the read lock for the data flash.	

Particularities and Limitations
> Function is only called when CRY_30_RH850ICUS_DATA_FLASH_SYNCHRONISATION is STD_ON.
Call context
> This function is called from task level only.

Table 5-47 Cry_30_Rh850Icus_DataFlashReadEnd_Callout

5.5.1.6 Cry_30_Rh850Icus_DataFlashWriteStart_Callout

Prototype	
`boolean` **`Cry_30_Rh850Icus_DataFlashWriteStart_Callout`** `(void)`	
Parameter	
-	
Return code	
`boolean`	TRUE if a write access to the data flash is granted, otherwise false.
Functional Description	
This function shall try to get a write lock for the data flash.	
Particularities and Limitations	
> Function is only called when CRY_30_RH850ICUS_DATA_FLASH_SYNCHRONISATION is STD_ON.	
Call context	
> This function is called from task level only.	

Table 5-48 Cry_30_Rh850Icus_DataFlashWriteStart_Callout

5.5.1.7 Cry_30_Rh850Icus_DataFlashWriteEnd_Callout

Prototype	
`void` **`Cry_30_Rh850Icus_DataFlashWriteEnd_Callout`** `(void)`	
Parameter	
-	
Return code	
-	
Functional Description	
This function shall try to release the write lock for the data flash.	
Particularities and Limitations	
> Function is only called when CRY_30_RH850ICUS_DATA_FLASH_SYNCHRONISATION is STD_ON.	
Call context	
> This function is called from task level only.	

Table 5-49 Cry_30_Rh850Icus_DataFlashWriteEnd_Callout

5.5.1.8 Cry_30_Rh850Icus_DataFlashSetReadMode_Callout

Prototype	
void **Cry_30_Rh850Icus_DataFlashSetReadMode_Callout** (void)	
Parameter	
-	
Return code	
-	
Functional Description	
If a command has been cancelled while it was updating a key, it may be necessary to switch the mode of the FACI Interface back to the read mode.	
Particularities and Limitations	
> Function is only called when CRY_30_RH850ICUS_DATA_FLASH_CONTROL is STD_ON.	
Call context	
> This function is called from task level only.	

Table 5-50 Cry_30_Rh850Icus_DataFlashSetReadMode_Callout

5.5.1.9 Cry_30_Rh850Icus_DataFlashReturnFromCommandLockedState_Callout

Prototype	
void **Cry_30_Rh850Icus_DataFlashReturnFromCommandLockedState_Callout** (void)	
Parameter	
-	
Return code	
-	
Functional Description	
Check if the Data lash is in the command locked state. If yes, bring it to a normal mode.	
Particularities and Limitations	
> Function is only called when CRY_30_RH850ICUS_DATA_FLASH_CONTROL is STD_ON.	
Call context	
> This function is called from task level only.	

Table 5-51 Cry_30_Rh850Icus_DataFlashReturnFromCommandLockedState_Callout

5.6 Services used by CRY_30_RH850ICUS

In the following table services provided by other components, which are used by the CRY_30_RH850ICUS are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
CSM	Csm_<Service>CallbackNotification Csm_<Service>ServiceFinishNotification

Table 5-52 Services used by the CRY_30_RH850ICUS

5.7 Service Ports

The current implementation of the CRY does not support Service Ports.

6 Configuration

In the CRY_30_RH850ICUS the attributes can be configured with the following tools:

> Configuration in DaVinci Configurator 5

6.1 Configuration Variants

The CRY_30_RH850ICUS supports the configuration variants

> `VARIANT-PRE-COMPILE`

6.2 Deviations

The current implementation does not have any deviations.

6.3 Additions/ Extensions

6.3.1 Timeout handling

The driver offers the possibility, to cancel a running command when it has consumed too much time. More information regarding this feature can be found in chapter 3.10.

6.3.2 Hardware error callout

The driver offers the possibility to send the hardware errors to a callout function. The callout function has to be implemented by the user (e.g. forwarding the error code). The API is described in chapter 5.5.1.3.

6.3.3 Data flash synchronization

If the data flash ressource is shared between the crypto hardware and the application core, a synchronization may be necessary to prevent race conditions. Therefore, the driver provides the feature to invoke callouts to request access to the data flash. The API is described in the chapter 5.5.1.4 - 5.5.1.7.

6.4 Limitations

6.4.1 Support of Cryptographic Services

The current cryptographic services are supported:

▶ SHE-AES128-Service for Symmetrical Interface
▶ SHE-PRNG-Service for Random Interface
▶ SHE-CMAC-Service for MAC Interface
▶ Service for Symmetrical Key Extract Interface
▶ Service for Symmetrical Key Wrapping Interface

Table 6-1 Supported AUTOSAR standard conform features

6.4.2 Parallel Access to Services

Due to limitations in the SHE, it’s not possible to process more than one service at once. An error (CSM_E_BUSY) is generated when a service is already running and another service tries to start at the same time.

Therefore parallel access to services which are dependent on the SHE is not allowed.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
Cryptographic Primitive	An underlying cryptographic module or library

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CRY	Cryptographic library module
CSM	Crypto Service Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SchM	Schedule Manager
SHE	Secure Hardware Extension
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com