MICROSAR CRYPTO

Technical Reference

CRYPTO VHSM

Version 4.0.1

Authors	vistof, visebj, viskju
Status	Released

Document Information

History

Author	Date	Version	Remarks
vistof	2017-12-15	1.00.00	Initial creation of Technical Reference
vistof	2017-02-15	1.01.00	Driver is now Hardware independent
vistof	2017-05-07	1.02.00	Release of component
visrpp	2018-09-24	2.00.00	Chapter 2.7.1, 2.7.4, 2.7.9 and 4.3 added.
visfpt	2018-09-25	2.00.00	Description of synchronization added
vistof	2018-10-10	2.00.00	Edit description of main function
vistof	2019-02-12	2.01.00	SafeBSW Release Chapter 4.2.3 added.
visjhi	2019-05-13	2.01.01	Version increment
vistof	2019-08-15	2.02.00	- Additional DET error - Rework of buffer handling - Add description for interrupt mode
vistof	2020-03-13	2.03.00	- Support for Async key APIs
vistof	2020-08-10	2.04.00	- Update supported features in chapter 2.1 - Add information for SafeBSW in chapter 0 - Add information about usage of the HOST2HSM register by the driver in chapter 2.3
vistof	2020-10-05	2.05.00	- Add optimization for streaming mode in chapter 2.8.6 - Remove limitation of parallel key management operations in chapter 2.1.3
vistof	2021-01-19	2.06.00	- Add information for NoBuffering in chapter 0 - Add missing Key Generate to list of supported features
vistof	2021-01-29	3.00.00	- IPCv3 - Add more detailed information to only provide buffers when needed in chapter 2.5.2
visebj	2021-07-26	3.01.00	- Add Set/Get for HSM status registers via KeyElement APIs in chapter 2.7.8
vistof	2022-10-25	3.03.00	Add Crypto_30_vHsm_RequestInterruptSet_Callout (4.3.5) Add Crypto_30_vHsm_ResponseInterruptClear_Callout (4.3.6) Add details for buffer handling in (0)
vistof	2023-01-24	3.03.01	Add IPC Channels (2.7.5)

viskju	2023-02-13	3.04.00	Added timeout callout (2.10, 4.3.2, 4.3.3, 4.3.4)
vistof	2023-05-26	4.00.00	Added Multi Partition support (2.1.3.5, 3.2, 2.11, 4.1.7)
vistof	2023-07-31	4.00.01	Add information for InitMemory() for Multi Partition (2.11.1)

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Crypto Driver	R4.3.0
[2]	AUTOSAR	Specification of Crypto Service Manager	R4.3.1
[3]	AUTOSAR	Specification of Default Error Tracer	R4.3.0
[4]	AUTOSAR	List of Basic Software Modules	R4.3.0
[5]	AUTOSAR	Specification of Crypto Driver	R4.4.0
[6]	AUTOSAR	Specification of Crypto Driver	R19-11
[7]	AUTOSAR	Specification of Crypto Driver	R20-11
[8]	Vector	Technical Reference of vHsmUpd_Ext_Vector	4.01.00

Contents

Illustrations

Tables

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CRYPTO as specified in [1].

Supported Configuration Variants:	pre-compile	
Vendor ID:	CRYPTO_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CRYPTO_MODULE_ID	114 decimal (according to ref. [4])

The Crypto Driver (CRYPTO) is called by the Crypto Interface (CRYIF) and performs the specific cryptographic functionality. The CRYPTO specification [1] offers a superset of algorithms which can be extended by ‘custom algorithms’. This software-based Crypto Driver offers a subset of algorithms and features which is described in 2.1.

1.1 Architecture Overview

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

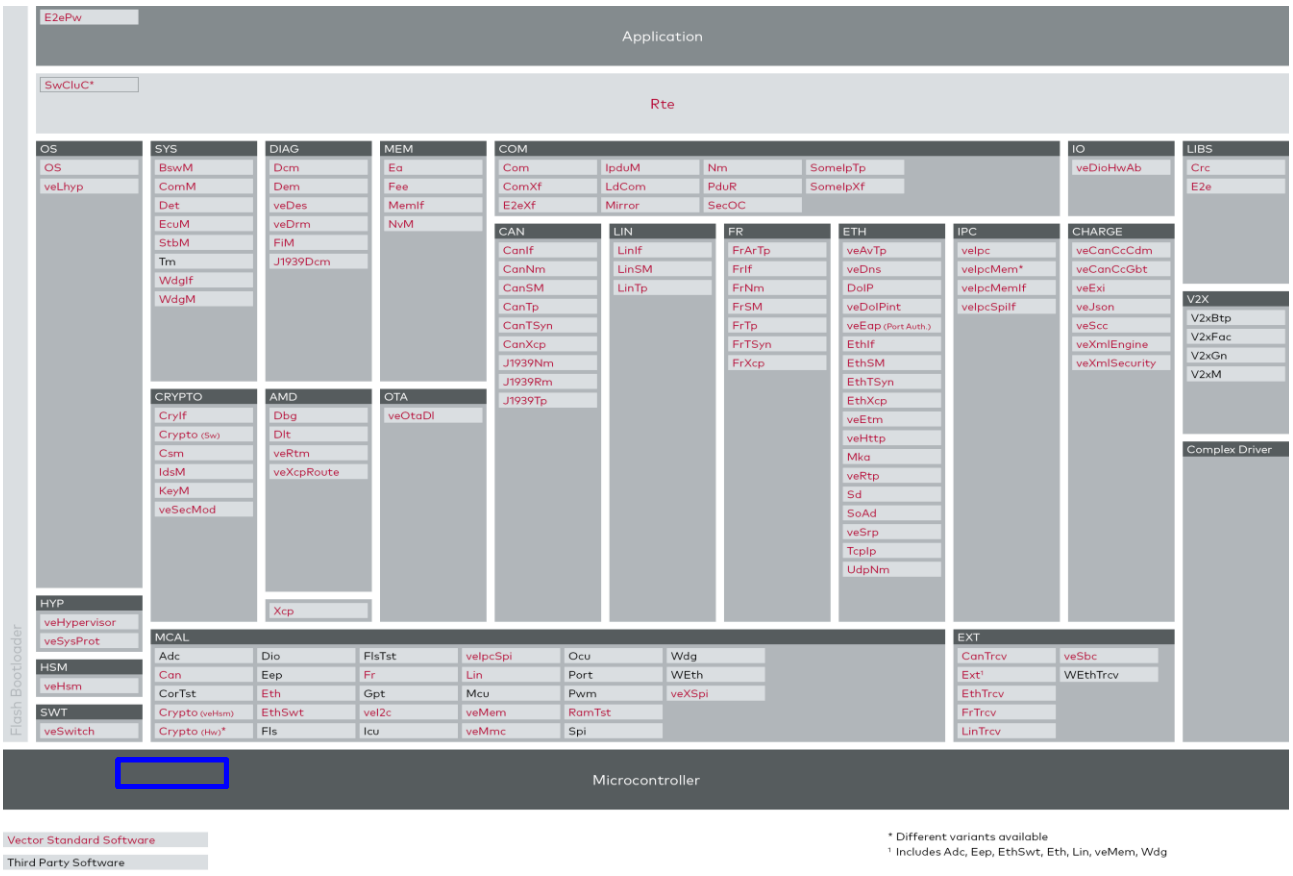

Figure 1-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the CRYPTO. These interfaces are described in chapter 4.

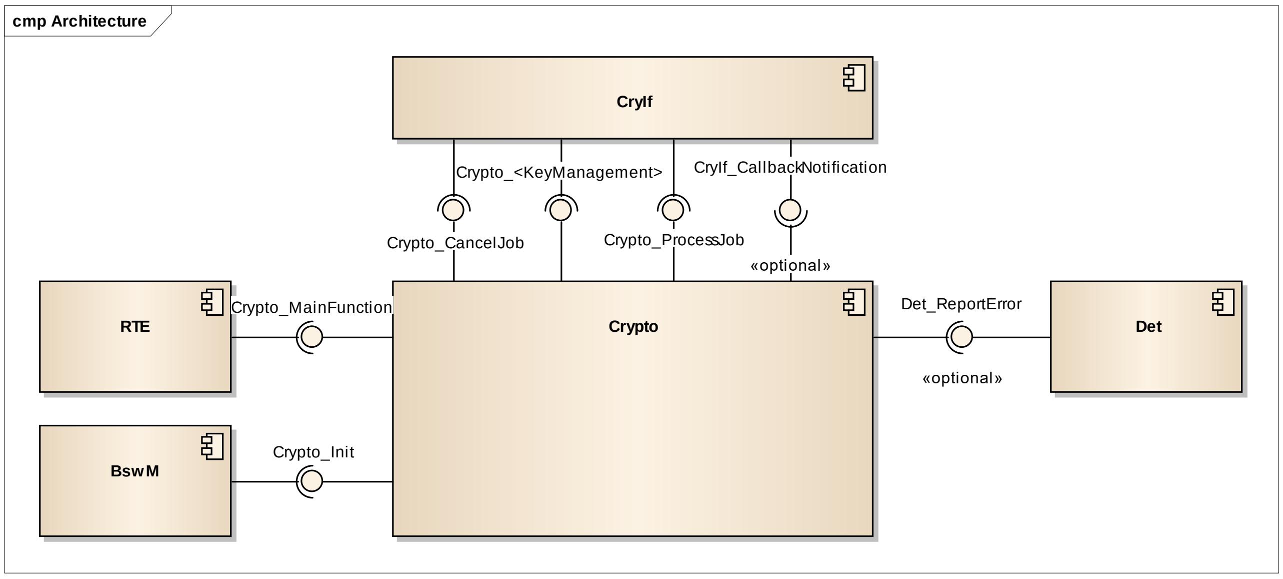

Figure 1-2 Interfaces to adjacent modules of the CRYPTO

2 Functional Description

2.1 Features

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

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

> Table 2-1 Supported AUTOSAR standard conform features

> Table 2-2 Not supported AUTOSAR standard conform features

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

> Table 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Setting and Getting of Keys and its elements
Key Copying
Random Seed
Key Derive
Key Generate
Key Exchange
Symmetric Job Execution
Asymmetric Job Execution
Several symmetric and asymmetric Primitives
Symmetric and asymmetric key generation
Cancelling of Jobs
Certificate handling
Key Management as jobs
Save and restore context

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

The following features specified in [1] are not supported:

Category	Description
API	Crypto_JobType is according to [2]
Config	Only 255 Driver objects are supported

Table 2-2 Not supported AUTOSAR standard conform features

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Retrieve version info from vHsm Firmware
Reading and setting reset vectors
Secure Boot
Flash operation synchronization
Secure Software Download
KeyElementCopyPartial specified in [5]
Redirection specified in [5]
Asynchronous key APIs [5]

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.3 Limitations

2.1.3.1 Cryptographic Algorithms and Modes

Only a subset of the stated algorithms and modes in the AUTOSAR SWS [1] are currently supported. The list of algorithms is described in Table 2-1.

2.1.3.2 Certificate Handling

Currently there is no implementation to parse and verify certificates. However, the API is still available for compatibility reasons.

2.1.3.3 AEAD Det Checks

The Det checks for AEAD differ from the table in [1]. There is a different handling required for the AEAD implementation. The parameter check is described in 2.5.2.

2.1.3.4 32 Bit Platform Support Only

Current implementation of the CRYPTO module is for 32 Bit platforms only. The usage with other platforms is prohibited.

2.1.3.5 Multi-partition

All partitions using the CRYPTO module have to use the same ASIL level.

2.2 Synchronization with vHsm

To synchronize the CRYPTO module with the vHsm, `Crypto_30_vHsm_WaitForHsmRam()` (or in the multi partition use case `Crypto_30_vHsm_WaitForHsmRamForApplication()`) can be called in the startup code. This function will wait until the vHsm finished all initialization tasks and is ready to receive requests. The synchronization is accomplished using a special value written to the IPC.

Depending on the hardware platform, there might be other ways for synchronization. E.g. communication using a register.

2.3 Initialization

Before any other functionality of the CRYPTO module (except `Crypto_30_vHsm_WaitForHsmRam()` or `Crypto_30_vHsm_WaitForHsmRamForApplication()`) can be called, the initialization function `Crypto_30_vHsm_Init()` must be called by the BSWM.

For manual null initialization of RAM variables, the CRYPTO offers the function `Crypto_30_vHsm_InitMemory()` which can be called before `Crypto_30_vHsm_Init()`. See chapter Memory Init (2.11.1) for more details in the Multi Partition Use-Case.

Expert Knowledge
If `/MICROSAR/Crypto_30_vHsm/Crypto/CryptovHsm/CryptoIpcInitialization` is enabled, the driver uses the HOST2HSM register to indicate to the HSM if the IPC has been initialized and usable.
Make sure that the register is set to 0 before the driver init is performed the first time by e.g. the bootloader. Multiple calls of the init function are possible as long as the above option is enabled.

Additionally, the driver will trigger an interrupt on HSM side to indicate an initialized IPC, if `/MICROSAR/Crypto_30_vHsm/Crypto/CryptovHsm/CryptoJobRequestInterrupt/CryptoJobRequestInterruptFlagSetting` is enabled.

Caution
If multiple calls of the driver init function are performed (e.g. Bootloader and afterwards application), make sure that all pending jobs are finished before calling the init function again.

It is also recommended to have `/MICROSAR/Crypto_30_vHsm/Crypto/CryptovHsm/CryptoIpcInitialization` enabled in this case.

2.4 Main Functions

The CRYPTO module implementation provides one main function. When the usage of asynchronous job processing is enabled, this main function has to be called cyclically on task level. The main function is responsible to poll the IPC for available job responses from the vHsm Firmware.

2.5 Error Handling

2.5.1 Development Error Reporting (DET)

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

The driver can also forward DET errors occurred inside the vHsm if the vHsm is configured accordingly.

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

The reported CRYPTO ID is 114.

The reported service IDs identify the services which are described in 4.1. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Crypto_30_vHsm_Init()
0x01	Crypto_30_vHsm_GetVersionInfo()
0x03	Crypto_30_vHsm_ProcessJob()
0x0E	Crypto_30_vHsm_CancelJob()
0x04	Crypto_30_vHsm_KeyElementSet()
0x05	Crypto_30_vHsm_KeyValidSet()
0x06	Crypto_30_vHsm_KeyElementGet()
0x0F	Crypto_30_vHsm_KeyElementCopy()
0x10	Crypto_30_vHsm_KeyCopy()
0x11	Crypto_30_vHsm_KeyElementIdsGet()
0x0D	Crypto_30_vHsm_RandomSeed()
0x07	Crypto_30_vHsm_KeyGenerate()
0x08	Crypto_30_vHsm_KeyDerive()
0x09	Crypto_30_vHsm_KeyExchangeCalcPubVal()
0x0A	Crypto_30_vHsm_KeyExchangeCalcSecret()
0x0B	Crypto_30_vHsm_CertificateParse()
0x12	Crypto_30_vHsm_CertificateVerify()
0x0C	Crypto_30_vHsm_MainFunction()
0x22	Crypto_30_vHsm_ProcessJobRequest()
0x21	Crypto_30_vHsm_GetResponseFromIpc()

Table 2-4 Service IDs

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

Error Code	Name	Description
0x00	CRYPTO_E_UNINIT	Module is not initialized
0x01	CRYPTO_E_INIT_FAILED	Error during init
0x02	CRYPTO_E_PARAM_POINTER	Passed pointer is invalid
0x04	CRYPTO_E_PARAM_HANDLE	Passed handle is invalid
0x05	CRYPTO_E_PARAM_VALUE	Passed value is invalid

Error Code	Name	Description
0x64	CRYPTO_E_GLOBAL_BUFFER_TOO_SMALL	The shared buffer configured in the driver object was too small to hold the result of a job

Table 2-5 Errors reported to DET

2.5.2 Algorithm Parameter Overview

The required algorithm parameters are described in the following table. The required parameter differs from the table in [1].

Member / Service	inputPtr	inputLength	secondaryInputPtr	secondaryInputLength	tertiaryInputPtr	tertiaryInputLength	outputPtr	outputLengthPtr	secondaryOutputPtr	secondaryOutputLengthPtr	verifyPtr	output64Ptr	mode
HASH	U	U					F	F					SUF
MACGENERATE	U	U					F	F					SUF
MACVERIFY	U	U	F	F							F		SUF
ENCRYPT	U	U					UF	UF					SUF
DECRYPT	U	U					UF	UF					SUF
AEADENCRYPT	U	U	V	U~			UF	UF	F	F			SUF
AEADDECRYPT	U	U	V	U~	F	F	UF	UF			F		SUF
SIGNATUREGENERATE	UF	U F~					F	F					SUF
SIGNATUREVERIFY	UF	U F~	F	F							F		SUF
SECCOUNTERINCREMENT													SUF
SECCOUNTERREAD												F	SUF
RANDOMGENERATE							F	F					SUF
RANDOMSEED	F	F											SUF
KEYGENERATE													SUF
KEYDERIVE													SUF
KEYEXCHANGE CALCPUBVAL							F	F					SUF
KEYEXCHANGE CALCPUBVAL	F	F											SUF
KEYSETVALID													SUF

S: member required in Start mode.
U: member required in Update mode.
V: member optional in Update mode.
F: member required in Finish mode.
~: no Det check required / 0 is a valid value.

Table 2-6 Overview of the required algorithm parameter

Caution
When Streaming mode is used and output pointers are provided which are not written by the vHsm, there will be unnecessary copy operations to the provided buffer with arbitrary data by the driver.

Do only provide buffers (input and output) when needed by the algorithm.

Example MAC Generate:

Do not provide the outputPtr for the mac during START or UPDATE call when streaming is used.

Otherwise the driver will copy arbitrary data to the provided outputPtr which is not an issue because the correct data is written during FINISH.

2.6 Key Usage

To simplify the key usage this chapter describes some basic aspects.

2.6.1 Custom Key Elements

There are custom key elements defined, which are shown in the following table.

Custom Key Element	Value
CRYPTO_KE_CUSTOM_MAC_AES_ROUNDKEY	129
CRYPTO_KE_CUSTOM_KEYDERIVATION_LABEL	130
CRYPTO_KE_CUSTOM_RSA_MODULUS	160
CRYPTO_KE_CUSTOM_RSA_PUBLIC_EXPONENT	161
CRYPTO_KE_CUSTOM_RSA_PRIVATE_EXPONENT	162
CRYPTO_KE_CUSTOM_RSA_SALT	163
CRYPTO_KE_CUSTOM_RSA_SALT_LENGTH	164
CRYPTO_KE_CUSTOM_TLS_CLIENT_HELLO_RANDOM	3000
CRYPTO_KE_CUSTOM_TLS_SERVER_HELLO_RANDOM	3001
CRYPTO_KE_CUSTOM_TLS_HMAC_KEY_SIZE	3002
CRYPTO_KE_CUSTOM_KEYEXCHANGE_PARTNER_PUB_KEY	3003
CRYPTO_KE_CUSTOM_SCC_CONTRACT_PUBLIC_KEY	3013
CRYPTO_KE_CUSTOM_SCC_IV_AND_ENCRYPTED_PRIVATE_KEY	3014
CRYPTO_KE_CUSTOM_RANDOM_PERSONALIZATION_STRING	3015
CRYPTO_KE_CUSTOM_RANDOM_ADDITIONAL_INPUT	3016
CRYPTO_KE_CUSTOM_RANDOM_NONCE	3017
CRYPTO_KE_CUSTOM_RANDOM_RESEED_COUNTER	3018
CRYPTO_KE_CUSTOM_SHE_COUNTER	3019
CRYPTO_KE_CUSTOM_SHE_UID	3021
CRYPTO_KE_CUSTOM_SHE_BOOT_PROTECTION	3056
CRYPTO_KE_CUSTOM_SHE_DEBUGGER_PROTECTION	3057
CRYPTO_KE_CUSTOM_SHE_DEBUG_CMD	3059
CRYPTO_KE_CUSTOM_RSA_PRIME_P	3051
CRYPTO_KE_CUSTOM_RSA_PRIME_Q	3052
CRYPTO_KE_CUSTOM_RSA_EXPONENT_DP	3053
CRYPTO_KE_CUSTOM_RSA_EXPONENT_DQ	3054
CRYPTO_KE_CUSTOM_RSA_INVERSE_QI	3055
CRYPTO_KE_CUSTOM_LABEL	3058
CRYPTO_KE_CUSTOM_VHSM_VERSION	3020
CRYPTO_KE_CUSTOM_VHSM_UID	3021
CRYPTO_KE_CUSTOM_VHSM_PERFORM_PERSISTING	3022
CRYPTO_KE_CUSTOM_VHSM_ERRORLOG	3023
CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_SLOT_ADDRESS	3024
CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_SLOT_SIZE	3025

CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_SLOT_CMAC	3026
CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_SLOT_KEY	1
CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_SLOT_SANCTION	3027
CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_END	3028
CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_RESULT	3037
CRYPTO_KE_CUSTOM_VHSM_BUNDLING_REQ	3029
CRYPTO_KE_CUSTOM_VHSM_BUNDLING_RESP	3030
CRYPTO_KE_CUSTOM_VHSM_BUNDLING_COUNTER	3031
CRYPTO_KE_CUSTOM_VHSM_FLASH_OPERATION	3032
CRYPTO_KE_CUSTOM_VHSM_RESET_VECTORS	3033
CRYPTO_KE_CUSTOM_VHSM_OPTION_BYTES	3034
CRYPTO_KE_CUSTOM_VHSM_UPGP	3035
CRYPTO_KE_CUSTOM_VHSM_PERFORM_REPERSISTING	3036
CRYPTO_KE_CUSTOM_RSA_PRIME_P	3051
CRYPTO_KE_CUSTOM_RSA_PRIME_Q	3052
CRYPTO_KE_CUSTOM_RSA_EXPONENT_DP	3053
CRYPTO_KE_CUSTOM_RSA_EXPONENT_DQ	3054
CRYPTO_KE_CUSTOM_RSA_INVERSE_QI	3055
CRYPTO_KE_CUSTOM_VHSM_STATUS_HSM2HOST	3090
CRYPTO_KE_CUSTOM_VHSM_STATUS_HOST2HSM	3091

Table 2-7 Custom Key Elements

2.6.2 Key Element Values

Some key elements only accept specific values, which are shown in the following table.

Key Element	Define	Value
CRYPTO_KE_CUSTOM_VHSM_FLASH_OPERATION	CRYPTO_30_VHSM_DATAFLASH_START	0x00
	CRYPTO_30_VHSM_DATAFLASH_STOP	0x01
	CRYPTO_30_VHSM_CODEFLASH_START	0x02
	CRYPTO_30_VHSM_CODEFLASH_STOP	0x03
CRYPTO_KE_KEYEXCHANGE_ALGOR	CRYPTO_KEY_EXCHANGE_X25519	0x00

	CRYPTO_KEY_EXCHANGE_ANSIP256R1	0x01
	CRYPTO_KEY_EXCHANGE_SECP256R1	0x2
	CRYPTO_KEY_EXCHANGE_SECP384R1	0x3
CRYPTO_KE_KEYDERIVATION_ALGORITHM	CRYPTO_KDF_ALGO_KDF_SYM_NIST_800_108_CNT_MODE_SHA256	0x01
	CRYPTO_KDF_ALGO_KDF_ASYM_NIST_FIPS_186_4_ERB	0x02
	CRYPTO_KDF_ALGO_KDF_NIST_800_56_A_ONE_PASS_C1E1S_SINGLE_STEP_KDF_SHA256	0x3
	CRYPTO_KDF_ALGO_KDF_ISO_15118_CERTIFICATE_HANDLING	0x04
	CRYPTO_KDF_ALGO_KDF_X963_SHA1	0x04
	CRYPTO_KDF_ALGO_KDF_X963_SHA256	0x5
	CRYPTO_KDF_ALGO_KDF_X963_SHA512	0x6
CRYPTO_KE_CUSTOM_VHSM_SECURE_BOOT_SLOT_SANCTION	CRYPTO_30_VHSM_SANCTION_NONE	0x01
	CRYPTO_30_VHSM_SANCTION_RESET	0x02
	CRYPTO_30_VHSM_SANCTION_CHANGE_RESET_VECTORS	0x03
	CRYPTO_30_VHSM_SANCTION_HALT	0x04

Table 2-8 Key Element Values

2.7 vHsm

This chapter describes specifics of the vHsm Firmware.

2.7.1 Pre Config File

The vHsm has the capability to generate a pre-config with the current settings of vHsm needed by CRYPTO, i.e. configured keys or driver objects including primitives.

This can be done for each IPC instance in the vHsm. The folder containing the pre-config file can be added to the list of additional definitions under the Project Settings of the application configuration.

Note

This Pre-Config mechanism may not work when a new driver is used with an old firmware version or vice versa.

Known conflicts:

1. If vHsm Firmware (<=R21) is used with Crypto Driver (>=R22), a replacement of strings inside the pre-config file is needed:

```
CRYPTO_ALGOFAM_CUSTOM_SOFTWARE_DOWNLOAD  -> CRYPTO_ALGOFAM_CUSTOM_VHSM_SOFTWARE_DOWNLOAD
CRYPTO_ALGOFAM_CUSTOM_SECURE_BOOT        -> CRYPTO_ALGOFAM_CUSTOM_VHSM_SECURE_BOOT
CRYPTO_ALGOFAM_CUSTOM_FIRMWARE_UPDATE    -> CRYPTO_ALGOFAM_CUSTOM_VHSM_FIRMWARE_UPDATE
CRYPTO_ALGOFAM_CUSTOM_SECURE_BOOT_UPDATE -> CRYPTO_ALGOFAM_CUSTOM_VHSM_SECURE_BOOT_UPDATE
```

2. If the driver is newer than the HSM, some new parameters may need to be configured manually because the HSM does not yet provide a recommendation for those parameters in the pre-config file.

2.7.2 Key Mapping

The key IDs on the Crypto driver and vHsm side must match each other.

To ensure this, use the generated pre-config file described in chapter 2.7.3 containing all accessible keys.

Note

vHsm <= R22:

The cryptoKeyId at the driver must match the cryptoKeyId in the LibCv driver inside the vHsm

vHsm > R22:

The cryptoKeyId in the driver must match the CryIfKeyId in the CryIf inside the vHsm.

2.7.3 Crypto Driver Objects

The Crypto Driver Objects, especially the ID, of CRYPTO and vHsm side must match each other.

To ensure this, use the generated pre-config file described in chapter 2.7.3 containing the driver objects including their provided primitives.

2.7.4 IPC Versions

The CRYPTO and the vHsm communicate over a RAM based IPC protocol. However, this protocol has a version which needs to be the same on both sides.

For HSM versions starting from Protocol Version 2, the version is synchronized via the generated pre-config file described in chapter 2.7.3.

For earlier versions, set this parameter manually to 1.

Protocol Version 1 only allows the usage of 4 driver objects of the vHsm:

- **Core:** Can be used to issue the secure boot and secure software update jobs
- **LibCv:** Can be used to compute algorithms inside the software implementation on the vHsm. The available primitives depend on the vHsm configuration and can be synchronized with a pre-config file generated by the vHsm.
- **Hal:** Can be used to compute algorithms in hardware such as CMAC, AES and TRNG
- **KeyM:** Cannot be used to perform jobs. This driver object is internally used to send the key management API calls to the vHsm. Do not reference this driver object in the CryIf.

With Protocol Version 2 and higher, there is no limit for driver objects anymore.

2.7.5 IPC Channels

Each IPC Instance consists of one or more channels where each channel can hold a job request (Key API calls like KeyElementSet are also counted as job requests). The parameter /MICROSAR/Crypto_30_vHsm/Crypto/CryptovHsm/CryptoIpcChannels defines the number of requests which can be processed at the same time without returning CRYPTO_E_BUSY. A channel is blocked until the response from the veHSM has been received.

Expert Knowledge

The number of channels configured in the veHSM must match the configuration in the Host driver. When it is known that for example more tasks will use the driver in future versions of the project, this must be considered because updating the veHsm without a working IPC communication is not possible without a debugger.

With R30 the veHSM supports multiple IPCs with different channel numbers. This can help in transitioning to an IPC with more channels in a system where an veHSM can't be updated with a debugger.

Caution
To avoid CRYPTO_E_BUSY because of blocked resources:

CryptoIpcChannels = Number of tasks which can interrupt each other and use Csm-Jobs or Key-APIs

CryptoManagementDriverObject references = Number of tasks which can interrupt each other and use Key-APIs

Caution
Not all upper layer modules like SWCs might implement a retry mechanism when CRYPTO_E_BUSY is returned.
Configuring the correct number of channels can avoid incompatibilities.

2.7.6 Memory Access

Input and output data for the vHsm must be located either in the code flash or shared RAM of the derivative. The vHsm cannot access the local RAM of the application cores or cached RAM on some platforms.

If the CRYPTO driver detects that the provided input or output memory location is not usable for the vHsm Firmware, it will copy the data to a buffer in the shared RAM and will provide the vHsm the pointer to the now usable memory location.

Basic Knowledge
The buffer of the Crypto_30_vHSM needs to be located in shared RAM. This needs to be ensured in the Memory Map!

The driver uses the CryptoMemoryAreas to decide if the shared buffer needs to be used.

Caution
If the vHsm shall be used in a safety use-case, it must be ensured that the HSM can only modify allowed memory areas. This can be achieved with an MPU. Refer to the corresponding hardware manual of the platform.

The driver on the host core must be configured to always provide output pointers pointing to the vHsmGlobalRamBuffer by having only READ areas configured and buffer size not equal to 0. With this the HSM can be restricted via MPU to only be able to have write access to vHsmIpcMemory and vHsmGlobalRamBuffer.

Expert Knowledge
A big buffer on the shared RAM and unnecessary copy operations can be avoided by ensuring that the data is in a valid input memory region. If it can be ensured that there is no buffering needed for a specific driver object, the buffer size can be set to 0 if there is not safety use-case and redirection is not used for that driver object.

Expert Knowledge
The container `/MICROSAR/Crypto_30_vHsm/Crypto/CryptoDriverObjects/CryptoDriverObject/CryptovHsmBuffer` is deprecated and the values inside are ignored. Pre-Config files from vHsm Firmwares < R23 will still contain the container which is why the definition contains the container for compatibility reasons.

Please use `/MICROSAR/Crypto_30_vHsm/Crypto/CryptoDriverObjects/CryptoDriverObject/CryptoBufferSize` to define the available size for the driver to copy not readable input and output data and provide the vHsm Firmware with valid pointers.

Expert Knowledge
If the configured buffer for the driver object cannot allocate enough space for input or output buffers, it will allocate as much as possible and will provide the reduced available buffer size to the vHsm firmware.
If the vHsm responds in such a case with CRYPTO_E_SMALL_BUFFER, a DET (if enabled) will be reported.

This is because upper layer modules might provide way to big buffers (e.g. 1k bytes) to store only a small amount of data (e.g. 16 bytes of CMAC). This would result in errors because the buffers in the driver objects are often configured smaller and could not allocate the big amount of memory space.

2.7.7 Special services

There might be further features like

- Getting version of vHsm Firmware
- Secure Boot
- vHsm Firmware Update
- Flash synchronization

Please refer to the corresponding technical references and the integration hints of the vHsm Firmware delivery see which features are available and how to use them.

2.7.8 Interrupt Mode

When the interrupt mode is enabled, make sure that the interrupt service routine `vHsmResponseIsr` is triggered by the interrupt issued from the vHsm Firmware. Synchronous jobs or key management APIs do not depend on an interrupt to occur to be processed correctly.

Expert Knowledge
On some Platforms (e.g. Tricore, Rcar) the OS will not clear the interrupt request flag.

To clear the flag inside the ISR configure the container `/MICROSAR/Crypto_30_vHsm/Crypto/CryptovHsm/CryptoInterruptHandling`

2.7.9 Status register

The status register for MICROSAR vHSM can be retrieved and set during runtime, by using the Crypto_30_vHsm_KeyElementGet and Crypto_30_vHsm_KeyElementSet APIs.

For this, the intended custom key elements for accessing the status registers must be used:

> CRYPTO_KE_CUSTOM_VHSM_STATUS_HSM2HOST

> CRYPTO_KE_CUSTOM_VHSM_STATUS_HOST2HSM

! **Set status register**
Please note that only the HOST2HSM status register can be written by CRYPTO.

2.7.9.1 Updater status information

If a vHsmUpd instance is active and running on the HSM core, this status information can be retrieved during runtime, by using the intended custom key element for reading the HSM2HOST register (`CRYPTO_KE_CUSTOM_VHSM_STATUS_HSM2HOST`).

vHsmUpd sets a corresponding bit in the HSM2HOST register see [8]. The bit number is defined within CRYPTO as `CRYPTO_30_VHSM_HSM2HOST_UPDATERRUNNING`.

2.8 Performance Improvements

There are different ways to improve the performance and throughput of jobs.

2.8.1 Use only necessary CryptoMemoryAreas

The performance of jobs depends on the number of memory sections. If it is possible, remove or disable memory sections which are never used as input or output. This can also be done in the vHsm Firmware configuration.

2.8.2 Sort CryptoMemoryAreas

Use priority to sort memory areas. Higher value equals higher priority means earlier checking. Areas which are accessed frequently should have a high priority.

2.8.3 Provide valid memory

Provide only data which is already in memory which can be directly used by the vHsm.

Depending on the platform, avoid e.g. local RAM of application cores as output

2.8.4 Provide aligned data

Provide pointers which are already aligned. Depending on the platform, the hardware accelerated algorithms need to align the data first before it can be passed to the hardware accelerator.

2.8.5 Use hardware acceleration of vHsm_Hal/Crypto_30_Hwa

Some algorithms can be computed with a hardware accelerator. Map jobs to HAL/HWA driver objects if possible. This can make huge performance improvements especially when there is a lot of data to be processed.

2.8.6 When job streaming is used: Provide buffers only during the corresponding call

If streaming mode is used and the buffer for e.g. CMAC will be filled in the finish step, only provide the output buffer for the CMAC in the FINISH call. Avoid passing the buffer for the output in START and UPDATE in this case, otherwise this will lead to unnecessary copy operations. Same applies for input pointers. SINGLECALL job calls are not affected by this.

2.8.7 Disable not needed critical sections

When hardware and compiler can guarantee atomic uint32 write operations the exclusive areas CRYPTO_30_VHSM_EXCLUSIVE_AREA_2 and CRYPTO_30_VHSM_EXCLUSIVE_AREA_3 can be empty.

2.8.8 Reduce Number of CryptoIpcChannels

Having not needed channels configured will decrease performance.

2.9 Wait Loop Callout

The CRYPTO has the configurable option to enable a callout which is called as long as the CRYPTO is waiting for the result of a synchronous job or key function.

The callout is described in chapter 4.3.1.

!

Caution
The callout function needs to be mapped to RAM in case the application uses vHsm features which require the HSM to access the code flash.

2.10 Timeout Detection Callouts

The CRYPTO has the configurable option to enable three callouts which are called respectively before, while and after the CRYPTO is waiting for a response of the HSM core (e.g. result of a synchronous job or key function).

This response can be delayed for a long time or never occur at all due to hardware or software defects. The callouts help to react on this situation. They can be used to detect violations of freedom from interference with respect to timing of the loops.

The detection and the reaction logic has to be implemented by the user.

To enable the callouts the configurable flag
`/MICROSAR/Crypto_30_vHsm/Crypto/CryptoGeneral/CryptoTimeoutCallout`
has to be set to true.

!

Caution
Currently there is no suitable reaction for too long running loops available other than performing a reset of the whole system including the HSM.

The timing of the wait loops is very hardware and project specific. Therefore, Vector cannot recommend suitable timeout values. As different loops also behave differently with respect to the timing, the callout receives multiple arguments, which can be used to differentiate between scenarios.

The `calloutId` parameter can have the following values:

Value	Description
CRYPTO_30_VHSM_CALLOUT_JOBREQUEST	Call is done when requesting the HSM to perform a synchronous job or key function.
CRYPTO_30_VHSM_CALLOUT_HSMREADY	Call is done when waiting for the HSM ready flag at boot up.
CRYPTO_30_VHSM_CALLOUT_IPCINIT	Call is done when waiting for the HSM acknowledge to IPC initialization at boot up.
CRYPTO_30_VHSM_CALLOUT_ENTERRAMLOOP	Call is done when waiting for the HSM to enter its RAM loop.
CRYPTO_30_VHSM_CALLOUT_EXITRAMLOOP	Call is done when waiting for the HSM to exit its RAM loop.
CRYPTO_30_VHSM_CALLOUT_ASYNC	Call is done when requesting or receiving an asynchronous job. The driver does not actively wait and therefore does never call the function `Crypto_30_vHsm_TimeoutDetect_Loop_Callout`.

Table 2-9 Possible values for calloutId

The rest of the parameters are only relevant if the `calloutId` parameter is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST or CRYPTO_30_VHSM_CALLOUT_ASYNC.

The parameter `objectId` can be used to adapt the callout behavior to the Id of the Crypto Driver Object (e.g. different behavior for software and hardware accelerated operations).

The parameter `contextClass` can have the following values:

Value	Description
CRYPTO_30_VHSM_CALLOUT_CRYPTOJOB	Current context is a crypto job.
CRYPTO_30_VHSM_CALLOUT_KEYMJOB	Current context is a key function.
CRYPTO_30_VHSM_CALLOUT_NOT_DEFINED	Current context is not defined.

Table 2-10 Possible values for contextClass

The parameter `contextId` has different meanings depending on the value of `contextClass`:

contextClass	contextId
CRYPTO_30_VHSM_CALLOUT_CRYPTOJOB	Id of the crypto job.
CRYPTO_30_VHSM_CALLOUT_KEYMJOB	Id of the key.

Table 2-11 Meaning of contextId

The callouts are described in chapter 4.3.2, 4.3.3 and 4.3.4.

Caution
The callout function needs to be mapped to RAM in case the application uses vHsm features which require the HSM to access the code flash.

2.11 Multi Partition Support

The CRYPTO supports being used on multiple partitions, which can be located on different cores. The accessing partitions have to be configured properly in `/MICROSAR/Crypto_30_vHsm/Crypto/CryptoGeneral/CryptoEcucPartitionRef`.

Caution
The user must ensure, that a partition only calls the Crypto using driver objects which were assigned to the accessing partition during configuration.

A check for this can be done by enabling `/MICROSAR/Crypto_30_vHsm/Crypto/CryptoGeneral/CryptoMultiPartitionRuntimeChecks`

Caution
There is additional information in the feature specific chapters which must be considered:

- 3.2 Critical Sections

Each partition uses an own IpcInstance to communicate with the veHsm. The Pre-config file contains all the needed information for the multi partition use-case if the veHsm is configured correctly. Dedicated variables and memory sections are created for each IpcInstance which need to be mapped correctly.

Caution
If hardware does not have dedicated registers to trigger interrupts to the HSM from the different partitions on different cores, make sure to take appropriate measures in the interrupt trigger callout like spinlocks or triggering an interrupt to the other core which then triggers an interrupt to the HSM or use the HSM in polling mode without interrupts. Same applies to clearing the response interrupt flags.

2.11.1 Memory Init for Multi Partition

The module generates dedicated `Crypto_30_vHsm_InitMemory_<OsApplicationName>()` functions for each partition which take care of initializing the partition dependent init variables to 0. Initializing can be done in the following ways:

- Startup core calls all available `Crypto_30_vHsm_InitMemory_<OsApplicationName>()` functions.
  - This requires that the startup core can write to the memory of all partitions.
  - This might require mapping the partition specific init variables to shared memory.
  - This would require dedicated MPU regions for the init variables of the partitions.
- Each core calls the `Crypto_30_vHsm_InitMemory_<OsApplicationName>()` functions for its partitions.
  - This can be used when init variables shall be mapped to core local memory which might not be accessible from other cores.
- Do not call the `Crypto_30_vHsm_InitMemory_<OsApplicationName>()` at all when ZERO_INIT variables are already correctly initialized.
  - Initialization is usually done by vLinkGen or Compiler specific startup code.

Expert Knowledge
Typically, `Crypto_30_vHsm_InitMemory()` is called before `OsStart()` so that MPU is not configured yet.

3 Integration

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

3.1 Scope of Delivery

The delivery of CRYPTO contains the files which are described in chapter 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Description
Crypto_30_vHsm.c	This is the main source file of the CRYPTO
Crypto_30_vHsm.h	This is the source file of the CRYPTO
Crypto_30_vHsm_Custom.h	This header contains driver specific custom defines for key elements and algorithm families or modes.
Crypto_30_vHsm_Ipc.c	This source file is used for the IPC to the vHsm
Crypto_30_vHsm_Ipc.h	This header file is used for the IPC to the vHsm
Crypto_30_vHsm_Ipc_Types.h	This header file is used for the IPC to the vHsm
Crypto_30_vHsm_Irq.c	This source file contains the interrupt service routine
Crypto_30_vHsm_Jobs.c	This source file is used for providing jobs to the vHsm
Crypto_30_vHsm_Jobs.h	This header file is used for providing jobs to the vHsm
Crypto_30_vHsm_KeyManagement.c	This source contains the CRYPTO´s key management functions
Crypto_30_vHsm_KeyManagement.h	This header contains the CRYPTO´s key management functions
Crypto_30_vHsm_TransformerLib.h	This header file is used for the read and write access to the IPC.
Crypto_30_vHsm_Types.h	This header contains types.

Table 3-1 Static files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator 5 Pro.

File Name	Description
Crypto_30_vHsm_Cfg.c	This is the configuration source file.
Crypto_30_vHsm_Cfg.h	This is the configuration header file.

Table 3-2 Generated files

3.2 Critical Sections

Crypto uses the following critical sections:

> CRYPTO_30_VHSM_EXCLUSIVE_AREA_0

This critical section protects global data from inconsistencies

> CRYPTO_30_VHSM_EXCLUSIVE_AREA_1

  This critical section ensures that a forwarded DET from the vHsm is processed only once (Flag is located in the IPC memory)

> CRYPTO_30_VHSM_EXCLUSIVE_AREA_2

  This critical section protects workspace locking resources when it cannot be ensured that uint32 write access is atomic. (see 2.8.7 “Disable not needed critical sections” for performance improvements)

> CRYPTO_30_VHSM_EXCLUSIVE_AREA_3

  This critical section protects workspace locking resources when it cannot be ensured that uint32 write access is atomic. If interrupt mode of driver is used and ISRs can't be interrupted by the same interrupt, this critical section does not need to be protected. (see 2.8.7 “Disable not needed critical sections” for performance improvements)

> CRYPTO_30_VHSM_EXCLUSIVE_AREA_4

> This critical section protects HOST2HSM register if multiple partitions are configured because multiple cores need to write to the register. This needs to be a spinlock for multi core configurations (see 2.8.7 “Disable not needed critical sections” for performance improvements)

See also comments of internal behavior in DaVinci Configurator for further information.

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Services provided by CRYPTO

4.1.1 Crypto_30_vHsm_Init

Prototype	
`void Crypto_30_vHsm_Init (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Initializes the Crypto Driver.	
Particularities and Limitations	
Specification of module initialization > Interrupts are disabled. Module is uninitialized. This function initializes the module Crypto_30_vHsm. It initializes all variables and sets the module state to initialized.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-1 Crypto_30_vHsm_Init

4.1.2 Crypto_30_vHsm_InitMemory

Prototype	
`void Crypto_30_vHsm_InitMemory (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
The function initializes variables, which cannot be initialized with the startup code.	
Particularities and Limitations	
Module is uninitialized. Initialize component variables at power up.	

Call context
> TASK > This function is Synchronous > This function is Non-Reentrant

Table 4-2 Crypto_30_vHsm_InitMemory

4.1.3 Crypto_30_vHsm_GetVersionInfo

Prototype	
`void` **`Crypto_30_vHsm_GetVersionInfo`** `(Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information.	
Particularities and Limitations	
Crypto_30_vHsm_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component.	
Call context	
> TASK|ISR > This function is Synchronous > This function is Reentrant	

Table 4-3 Crypto_30_vHsm_GetVersionInfo

4.1.4 Crypto_30_vHsm_ProcessJob

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_ProcessJob`** `(uint32 objectId, Crypto_JobType *job)`	
Parameter	
objectId [in]	Holds the identifier of the Crypto Driver Object.
job [in,out]	Pointer to the configuration of the job. Contains structures with job and primitive relevant information but also pointer to result buffers.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.

	CRYPTO_E_KEY_NOT_VALID Request failed, the key is not valid.
	CRYPTO_E_QUEUE_FULL Request failed, the queue is full.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Process the received job.	
Particularities and Limitations	
Performs the crypto primitive that is configured in the job parameter.	
Call context	
> TASK > This function is Reentrant	

Table 4-4 Crypto_30_vHsm_ProcessJob

4.1.5 Crypto_30_vHsm_CancelJob

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_CancelJob`** `(uint32 objectId, Crypto_JobType *job)`	
Parameter	
objectId [in]	Holds the identifier of the Crypto Driver Object.
job [in,out]	Pointer to the configuration of the job. Contains structures with user and primitive relevant information.
Return code	
Std_ReturnType	E_OK Request successful, job has been removed.
Std_ReturnType	E_NOT_OK Request failed, job could not be removed.
Functional Description	
Cancels the received job.	
Particularities and Limitations	
This interface removes the provided job from the queue and cancels the processing of the job if possible.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-5 Crypto_30_vHsm_CancelJob

4.1.6 Crypto_30_WaitForHsmRam

Prototype
`Void` **`Crypto_30_vHsm_WaitForHsmRam`** `(void)`

Parameter	
None	
Return code	
None	
Functional Description	
Waits until vHsm is ready after reset.	
Particularities and Limitations	
This function is mapped to RAM to avoid conflicting access to code flash with the vHsm. It waits for the ready flag in the HSM2HOST register and also if the 0xDEADBEEF pattern is written to the IPC. Can be called before Crypto_30_vHsm_Init(). Can be called from Startup Code. Is only available for single partition use case.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 4-6 Crypto_30_vHsm_WaitForHsmRam

4.1.7 Crypto_30_WaitForHsmRamForApplication

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_WaitForHsmRamForApplication`** `(ApplicationType applicationId)`	
Parameter	
applicationId	Valid ApplicationId which is used to select which IPC is checked and initialized
Return code	
Std_ReturnType	E_OK Request successful, job has been removed.
Std_ReturnType	E_NOT_OK Request failed, job could not be removed.
Functional Description	
Waits until vHsm is ready after reset.	
Particularities and Limitations	
This function is mapped to RAM to avoid conflicting access to code flash with the vHsm. It waits for the ready flag in the HSM2HOST register and also if the 0xDEADBEEF pattern is written to the IPC. Can be called before Crypto_30_vHsm_Init(). Can be called from Startup Code. Is only available for multi partition use case. Consider the fact that GetApplicationId() from Os can not be called during startup code to get the applicationId automatically and therefore manual integration is needed.	

Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 4-7 Crypto_30_vHsm_WaitForHsmRamForApplication

4.2 Key Management Functions

4.2.1 Crypto_30_vHsm_KeyCopy

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyCopy`** `(uint32 cryptoKeyId, uint32 targetCryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be the source element.
targetCryptoKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_READ_FAIL Request failed, read access was denied.
	CRYPTO_E_KEY_WRITE_FAIL Request failed, write access was denied.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element sizes are not compatible.
Functional Description	
Copy the key.	
Particularities and Limitations	
Copies a key with all its elements to another key in the same crypto driver. The key element can only be copied, if the destination key element write access right is less than WA_INTERNAL_COPY. Additional, the read access right of the source must be less than RA_INTERNAL_COPY and the destination read access right must be higher or equal than the source read access right.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-8 Crypto_30_vHsm_KeyCopy

4.2.2 Crypto_30_vHsm_KeyElementCopy

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyElementCopy`** `(uint32 cryptoKeyId, uint32 keyElementId, uint32 targetCryptoKeyId, uint32 targetKeyElementId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be the source element.
keyElementId [in]	Holds the identifier of the key element which shall be the source for the copy operation.
targetCryptoKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
targetKeyElementId [in]	Holds the identifier of the key element which shall be the destination for the copy operation.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_READ_FAIL Request failed, read access was denied.
	CRYPTO_E_KEY_WRITE_FAIL Request failed, write access was denied.
	CRYPTO_E_KEY_EXTRACT_DENIED Request failed, not allowed to extract key material.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element sizes are not compatible.
Functional Description	
Copy key element.	
Particularities and Limitations	
Copies a key element to another key element in the same crypto driver. The key element can only be copied, if the destination key element write access right is less than WA_INTERNAL_COPY. Additional, the read access right of the source must be less than RA_INTERNAL_COPY and the destination read access right must be higher or equal than the source read access right.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-9 Crypto_30_vHsm_KeyElementCopy

4.2.3 Crypto_30_vHsm_KeyElementCopyPartial

Prototype
`Std_ReturnType` **`Crypto_30_vHsm_KeyElementCopyPartial`** `(uint32 cryptoKeyId,`

uint32 keyElementId, uint32 keyElementSourceOffset, uint32 keyElementTargetOffset, uint32 keyElementCopyLength, uint32 targetCryptoKeyId, uint32 targetKeyElementId)	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be the source element.
keyElementId [in]	Holds the identifier of the key element which shall be the source for the copy operation.
keyElementSourceOffset [in]	Holds the offset of the of the source key element indicating the start index of the copy operation.
keyElementTargetOffset [in]	Holds the offset of the of the target key element indicating the start index of the copy operation.
keyElementCopyLength [in]	Holds the number of bytes that shall be copied.
targetCryptoKeyId [in]	Holds the identifier of the key whose key element shall be the destination element.
targetKeyElementId [in]	Holds the identifier of the key element which shall be the destination for the copy operation.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_READ_FAIL Request failed, read access was denied.
	CRYPTO_E_KEY_WRITE_FAIL Request failed, write access was denied.
	CRYPTO_E_KEY_EXTRACT_DENIED Request failed, not allowed to extract key material.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element sizes are not compatible.
	CRYPTO_E_KEY_EMPTY Request failed, uninitialized source key element.
Functional Description	
Copy key element partial. Copies a key element to another key element in the same crypto driver. The keyElementSourceOffset and keyElementCopyLength allows to copy just a part of the source key element into the destination. The offset of the target key is also specified with this function. The target key element needs to have partial access. The key element can only be copied, if the destination key element write access right is less than WA_INTERNAL_COPY. Additional the read access right of the source must be less than RA_INTERNAL_COPY and the destination read access right must be higher or equal than the source read access right.	
Particularities and Limitations	
-	
Call context	
> TASK	

> This function is Synchronous > This function is Reentrant for different crypto keys.

Table 4-10 Crypto_30_vHsm_KeyElementCopyPartial

4.2.4 Crypto_30_vHsm_KeyElementIdsGet

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyElementIdsGet`** `(uint32 cryptoKeyId, uint32 *keyElementIdsPtr, uint32 *keyElementIdsLengthPtr)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose available element ids shall be exported.
keyElementIdsLengthPtr [in]	Holds a pointer to the memory location in which the number of key element in the given key is stored. On calling this function, this parameter shall contain the size of the buffer provided by keyElementIdsPtr. When the request has finished, the actual number of key elements is stored.
keyElementIdsPtr [out]	Contains the pointer to the array where the ids of the key elements shall be stored.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Used to retrieve information which key elements are available in a given key.	
Particularities and Limitations	
none	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-11 Crypto_30_vHsm_KeyElementIdsGet

4.2.5 Crypto_30_vHsm_KeyElementSet

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyElementSet`** `(uint32 cryptoKeyId, uint32 keyElementId, const uint8 *keyPtr, uint32 keyLength)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be set.
keyElementId [in]	Holds the identifier of the key element which shall be set.

keyPtr [in]	Holds the pointer to the key data which shall be set as key element.
keyLength [in]	Contains the length of the key element in bytes.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_WRITE_FAIL Request failed, write access was denied.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the key element size does not match size of provided data.
Functional Description	
Sets a key element.	
Particularities and Limitations	
Sets the given key element bytes to the key identified by cryptoKeyId. The key element can only be set, if the write access right is WA_ALLOWED or WA_ENCRYPTED for she key update.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-12 Crypto_30_vHsm_KeyElementSet

4.2.6 Crypto_30_vHsm_KeyValidSet

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyValidSet`** `(uint32 cryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key elements shall be set to valid.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Sets the key to valid.	
Particularities and Limitations	
Sets the key state of the key identified by cryptoKeyId to valid.	
Call context	
> TASK > This function is Synchronous	

> This function is Reentrant for different crypto keys.

Table 4-13 Crypto_30_vHsm_KeyValidSet

4.2.7 Crypto_30_vHsm_KeyElementGet

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyElementGet`** `(uint32 cryptoKeyId, uint32 keyElementId, uint8 *resultPtr, uint32 *resultLengthPtr)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key whose key element shall be set.
keyElementId [in]	Holds the identifier of the key element which shall be set.
resultPtr [in]	Holds the pointer to the key data which shall be set as key element.
resultLengthPtr [in]	Contains the length of the key element in bytes.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_KEY_READ_FAIL Request failed, read access was denied.
	CRYPTO_E_KEY_NOT_AVAILABLE Request failed, the key is not available.
	CRYPTO_E_KEY_SIZE_MISMATCH Request failed, the provided buffer is too small to store the result.
Functional Description	
This interface shall be used to get a key element of the key identified by the cryptoKeyId and store the key element in the memory location pointed by the result pointer.	
Particularities and Limitations	
The key element can only be provided, if the read access right is RA_ALLOWED.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-14 Crypto_30_vHsm_KeyElementGet

4.2.8 Crypto_30_vHsm_RandomSeed

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_RandomSeed`** `(uint32 cryptoKeyId, const uint8 *entropyPtr, uint32 entropyLength)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key for which a new seed shall be generated.

entropyPtr [in]	Holds a pointer to the memory location which contains the data to feed the entropy.
entropyLength [in]	Contains the length of the entropy in bytes.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Initialize the seed.	
Particularities and Limitations	
This function generates the internal seed state using the provided entropy source. Furthermore, this function can be used to update the seed state with new entropy.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-15 Crypto_30_vHsm_RandomSeed

4.2.9 Crypto_30_vHsm_KeyGenerate

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyGenerate`** `(uint32 cryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which is to be updated with the generated value.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Generates a key.	
Particularities and Limitations	
This function shall dispatch the key generate function to the configured crypto driver object.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-16 Crypto_30_vHsm_KeyGenerate

4.2.10 Crypto_30_vHsm_KeyDerive

Prototype	
`Std_ReturnType **Crypto_30_vHsm_KeyDerive** (uint32 cryptoKeyId, uint32 targetCryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which is used for key derivation.
targetCryptoKeyId [in]	Holds the identifier of the key which is used to store the derived key.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Derives a key.	
Particularities and Limitations	
Derives a new key by using the key elements in the given key identified by the cryptoKeyId. The given key must contain the key elements for the password (CRYPTO_KE_KEYDERIVATION_PASSWORD), salt (CRYPTO_KE_KEYDERIVATION_SALT), algorithm (CRYPTO_KE_KEYDERIVATION_ALGORITHM) and the custom element label(CRYPTO_KE_CUSTOM_KEYDERIVATION_LABEL). The number of iterations is automatically determined by the needed key length. The derived key is stored in the key element with id 1 of the given key identified by targetCryptoKeyId. The write access right for the destination key element must be less or equal than WA_INTERNAL_COPY.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-17 Crypto_30_vHsm_KeyDerive

4.2.11 Crypto_30_vHsm_KeyExchangeCalcPubVal

Prototype	
`Std_ReturnType **Crypto_30_vHsm_KeyExchangeCalcPubVal** (uint32 cryptoKeyId, uint8 *publicValuePtr, uint32 *publicValueLengthPtr)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which shall be used for the key exchange protocol.
publicValuePtr [out]	Contains the pointer to the data where the public value shall be stored.
publicValueLengthPtr [in,out]	Holds a pointer to the memory location in which the public value length information is stored. On calling this function, this parameter shall contain the

	size of the buffer provided by publicValuePtr. When the request has finished, the actual length of the returned value shall be stored.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Calculation of the public value.	
Particularities and Limitations	
Calculates the public value for the key exchange and stores the public key in the memory location pointed by the public value pointer. The write access right for the destination key element must be less or equal than WA_INTERNAL_COPY.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-18 Crypto_30_vHsm_KeyExchangeCalcPubVal

4.2.12 Crypto_30_vHsm_KeyExchangeCalcSecret

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_KeyExchangeCalcSecret`** `(uint32 cryptoKeyId, const uint8 *partnerPublicValuePtr, uint32 partnerPublicValueLength)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which shall be used for the key exchange protocol.
partnerPublicValuePtr [in]	Holds the pointer to the memory location which contains the partners public value.
partnerPublicValueLength [in]	Contains the length of the partners public value in bytes.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
	CRYPTO_E_SMALL_BUFFER Request failed, the provided buffer is too small to store the result.
Functional Description	
Calculation of the secret.	

Particularities and Limitations
Calculates the shared secret key for the key exchange with the key material of the key identified by the cryptoKeyId and the partner public key. The shared secret key is stored as a key element in the same key. The write access right for the destination key element must be less or equal than WA_INTERNAL_COPY.
Call context
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.

Table 4-19 Crypto_30_vHsm_KeyExchangeCalcSecret

4.2.13 Crypto_30_vHsm_CertificateParse

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_CertificateParse`** `(uint32 cryptoKeyId)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key slot in which the certificate has been stored.
Return code	
Std_ReturnType	E_OK Request successful.
	E_NOT_OK Request failed.
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.
Functional Description	
Parse stored certificate.	
Particularities and Limitations	
Parses the certificate data stored in the key element CRYPTO_KE_CERT_DATA and fills the key elements CRYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_CERT_SIGNATURE	
Call context	
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.	

Table 4-20 Crypto_30_vHsm_CertificateParse

4.2.14 Crypto_30_vHsm_CertificateVerify

Prototype	
`Std_ReturnType` **`Crypto_30_vHsm_CertificateVerify`** `(uint32 cryptoKeyId, uint32 verifyCryptoKeyId, Crypto_VerifyResultType *verifyPtr)`	
Parameter	
cryptoKeyId [in]	Holds the identifier of the key which shall be used to validate the certificate.

verifyCryptoKeyId [in]	Holds the identifier of the key containing the certificate, which shall be verified.	
verifyPtr [out]	Holds a pointer to the memory location which will contain the result of the certificate verification.	
Return code		
Std_ReturnType	E_OK Request successful.	
	E_NOT_OK Request failed.	
	CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy.	
Functional Description		
Certificate verification.		
Particularities and Limitations		
Verifies the certificate stored in the key referenced by verifyCryptoKeyId with the certificate stored in the key referenced by cryptoKeyId.		
Call context		
> TASK > This function is Synchronous > This function is Reentrant for different crypto keys.		

Table 4-21 Crypto_30_vHsm_CertificateVerify

4.3 Callout Functions

CRYPTO defines some configurable callout functions. The declarations of the callout functions are provided by the CRYPTO module. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The CRYPTO callout function declarations are described in the following tables:

4.3.1 Wait Loop Callout

Prototype	
`void` **`<WaitLoopCalloutName>`** `(void)`	
Parameter	
`void`	none
Return code	
`void`	none
Functional Description	
Notification about ongoing waiting for a response from the vHsm Firmware for a synchronous job.	
Particularities and Limitations	
> Implementation has to be done in the application. > The name of this function is configurable. > If HSM is required to execute code flash operations, this functions needs to be mapped to RAM.	

Call context
> Interrupt or task context > Callout is invoked whenever driver is waiting for a synchronous job > This function is Synchronous > This function is Reentrant

Table 4-22 Wait Loop Callout

4.3.2 Crypto_30_vHsm_TimeoutDetect_Start_Callout

Prototype	
void **Crypto_30_vHsm_TimeoutDetect_Start_Callout**(Crypto_30_vHsm_CalloutIdType calloutId, uint32 objectId, Crypto_30_vHsm_CalloutContextClassType contextClass, uint32 contextId)	
Parameter	
`calloutId`	Identifies the loop.
`objectId`	Id of the Crypto Driver Object of the current context. This is only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST or CRYPTO_30_VHSM_CALLOUT_ASYNC.
`contextClass`	Type of the current context. Only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST or CRYPTO_30_VHSM_CALLOUT_ASYNC.
`contextId`	If contextClass is CRYPTO_30_VHSM_CALLOUT_CRYPTOJOB, this parameter is the crypto job id, if contextClass is CRYPTO_30_VHSM_CALLOUT_KEYMJOB, this parameter is the key id. Only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST.
Return code	
`void`	none
Functional Description	
Notification just before waiting for a response from the vHsm Firmware.	
Particularities and Limitations	
> Implementation has to be done in the application. > If HSM is required to execute code flash operations, this functions needs to be mapped to RAM.	
Call context	
> Interrupt or task context > Callout is invoked whenever driver is waiting for HSM Firmware. > This function is Synchronous > This function is Reentrant	

Table 4-23 Crypto_30_vHsm_TimeoutDetect_Start_Callout

4.3.3 Crypto_30_vHsm_TimeoutDetect_Loop_Callout

Prototype	
`void Crypto_30_vHsm_TimeoutDetect_Loop_Callout (Crypto_30_vHsm_CalloutIdType calloutId, uint32 objectId, Crypto_30_vHsm_CalloutContextClassType contextClass, uint32 contextId)`	
Parameter	
`calloutId`	Identifies the loop.
`objectId`	Id of the Crypto Driver Object of the current context. This is only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST or CRYPTO_30_VHSM_CALLOUT_ASYNC.
`contextClass`	Type of the current context. Only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST or CRYPTO_30_VHSM_CALLOUT_ASYNC.
`contextId`	If contextClass is CRYPTO_30_VHSM_CALLOUT_CRYPTOJOB, this parameter is the crypto job id, if contextClass is CRYPTO_30_VHSM_CALLOUT_KEYMJOB, this parameter is the key id. Only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST.
Return code	
`void`	none
Functional Description	
Notification about ongoing waiting for a response from the vHsm Firmware.	
Particularities and Limitations	
> Implementation has to be done in the application. > If HSM is required to execute code flash operations, this functions needs to be mapped to RAM.	
Call context	
> Interrupt or task context > Callout is invoked whenever driver is waiting for HSM Firmware. > This function is Synchronous > This function is Reentrant	

Table 4-24 Crypto_30_vHsm_TimeoutDetect_Loop_Callout

4.3.4 Crypto_30_vHsm_TimeoutDetect_End_Callout

Prototype	
`void Crypto_30_vHsm_TimeoutDetect_End_Callout(Crypto_30_vHsm_CalloutIdType calloutId, uint32 objectId, Crypto_30_vHsm_CalloutContextClassType contextClass, uint32 contextId)`	
Parameter	
`calloutId`	Identifies the loop.

`objectId`	Id of the Crypto Driver Object of the current context. This is only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST or CRYPTO_30_VHSM_CALLOUT_ASYNC.
`contextClass`	Type of the current context. Only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST or CRYPTO_30_VHSM_CALLOUT_ASYNC.
`contextId`	If contextClass is CRYPTO_30_VHSM_CALLOUT_CRYPTOJOB, this parameter is the crypto job id, if contextClass is CRYPTO_30_VHSM_CALLOUT_KEYMJOB, this parameter is the key id. Only relevant if calloutId is CRYPTO_30_VHSM_CALLOUT_JOBREQUEST.
Return code	
`void`	none
Functional Description	
Notification just after waiting for a response from the vHsm Firmware.	
Particularities and Limitations	
> Implementation has to be done in the application. > If HSM is required to execute code flash operations, this functions needs to be mapped to RAM.	
Call context	
> Interrupt or task context > Callout is invoked whenever driver is waiting for HSM Firmware. > This function is Synchronous > This function is Reentrant	

Table 4-25 Crypto_30_vHsm_TimeoutDetect_End_Callout

4.3.5 Crypto_30_vHsm_RequestInterruptSet_Callout

Prototype	
`void` **`Crypto_30_vHsm_RequestInterruptSet_Callout`**`(uint32 address, uint32 mask)`	
Parameter	
`address`	Address configured in the Cfg5 for the request interrupt setting
`mask`	Mask configured in the Cfg5 for the request interrupt setting
Return code	
`void`	none
Functional Description	
Function which is called to set the request interrupt flag.	
Particularities and Limitations	
Usually the driver provides the capability to handle setting the interrupt flag by itself. However new platforms or use-cases may require more specific handling which can be implemented in this callout.	

Call context
> Interrupt or task context > This function is Synchronous > This function is Reentrant

Table 4-26 Crypto_30_vHsm_RequestInterruptSet_Callout

4.3.6 Crypto_30_vHsm_ResponseInterruptClear_Callout

Prototype	
`void` **`Crypto_30_vHsm_ResponseInterruptClear_Callout`**`(uint32 address, uint32 mask, uint32 value)`	
Parameter	
`address`	Address configured in the Cfg5 for the response interrupt clearing
`mask`	Mask configured in the Cfg5 for the response interrupt clearing
`value`	Value configured in the Cfg5 for the response interrupt clearing
Return code	
`void`	none
Functional Description	
Function which is called to clear the reponse interrupt flag.	
Particularities and Limitations	
Usually the driver provides the capability to handle clearing the interrupt flag by itself. However new platforms or use-cases may require more specific handling which can be implemented in this callout.	
Call context	
> Interrupt or task context > This function is Synchronous > This function is Reentrant	

Table 4-27 Crypto_30_vHsm_ResponseInterruptClear_Callout

4.4 Services needed by CRYPTO

No services are needed for CRYPTO.

4.5 Services used by CRYPTO

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

Component	API
DET	Det_ReportError
CRYIF	CryIf_CallbackNotification

Table 4-28 Services used by the CRYPTO

5 Configuration

In the CRYPTO the attributes can be configured according to/ with the following methods/ tools:

> Configuration in DaVinci Configurator 5 Pro

5.1 Configuration Variants

The CRYPTO supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the CRYPTO parameters depend on the supported configuration variants. For their definitions please see the Crypto_30_vHsm_bswmd.arxml file.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
CSM	Crypto Service Manager
CRYIF	Crypto Interface
CRYPTO	Crypto Driver

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PPORT	Provide Port
RPORT	Require Port
RTE	Runtime Environment
SHE	Secure Hardware Extension
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com