VECTOR

Security Module Basic

Technical Reference

Version 2.2.0

Authors	Markus Schneider
Status	Released

Document Information

History

Author	Date	Version	Remarks
Florian Hees	2007-03-12	1.0.0	Initial creation
Markus Schneider	2013-11-27	2.1.0	Major rework
André Caspari	2016-08-02	2.2.0	Updated API description

Reference Documents

No.	Source	Title	Version
[1]	HIS	API IO Library	2.0.3
[2]	HIS	Security Module Specification	1.1
[3]	Vector	HexView Reference Manual	1.6

Scope of the Document

This technical reference describes the general use of the Security Module software.

Contents

Illustrations

Tables

1 Introduction

The Bootloader comes along with a basic security module that supports the security class DDD (CRC-32). This document describes the functions and the configuration of this module. The security classes are standardized by the HIS (see [2]).

The security module can be found in the folder \BSW\SecMod.

This document covers only the basic security module. The Bootloader can also be ordered with security modules for the security classes C, CCC and AAA, depending on the contracts and level of purchases. These security modules are described in another document.

2 Software Architecture

The Security Module delivery contains the following components:

> The security module itself (with its sub-packages)

> Configuration tool GENy

In addition, the following tool is delivered:

> The software tool HexView. It can be used to change and show the contents of a binary file. Also it can be used for signature and checksum calculation. Besides, the creation of encrypted download files is possible with HexView.

2.1 Security Module Components

The functionalities of the Security Module are separated into several sub-packages. A functional description for the individual modules can be found in chapter 6.

Depending on your order the following modules can be provided with the Security Module:

Name	Description
CRC	Implementation of the HIS security module - CRC calculation. Offers CRC calculation according to CRC-16 CCITT or CRC-32 (IEEE 802.3).
SeedKey	Implementation of the HIS security module - Seed/key authentication. Offers challenge/response authentication interface.
Verification	Implementation of the verification component of the HIS security module - Signature verification. Offers signature/checksum verification interface.
Sec	Implementation of the HIS security module.

Table 2-1 Security Module Components

3 Integration

This chapter gives necessary information for the integration of the Vector Security Module into an application environment of an ECU.

3.1 Scope of Delivery

The delivery of the Security Module contains the following files:

Module	Filename	Delivery Type		Description
		Source	Lib	
CRC	Sec_Crc.c	■		Source file of the CRC Module.
	Sec_Crc.h	■		Header file of the CRC Module.
SeedKey	Sec_SeedKey.c	■		Source file of the Seed/Key Module.
	Sec_SeedKey.h	■		Header file of the Seed/Key Module.
Verification	Sec_Verification.c	■		Source file of the Verification Module.
	Sec_Verification.h	■		Header file of the Verification Module.
Common	Sec_Inc.h	■		Header file used internally.
	Sec_Types.h	■		Defines types, constantans and configuration switches.
	Sec.c	■		Source file of the Security Module.
	Sec.h	■		Header file of the Security Module.

Table 3-1 Scope of Delivery

3.2 Include Structure

For usage of the Security Module within an application it is necessary to include the ‘Sec.h’ header file.

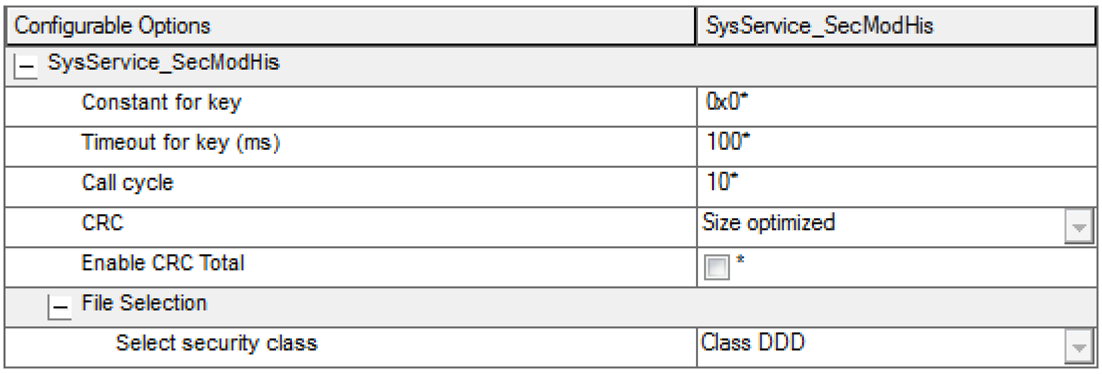
4 Configuration

4.1 GENy

We recommend configuring the Security Module with the configuration tool GENy. Depending on your order the tool can be downloaded from our FTP and installed.

4.2 Security Module Configuration

The features of the security component are configured in the 'SysService_SecModHis' component. The available features or pre-set features are shown on the right side of the window.



Configurable Options	SysService_SecModHis
SysService_SecModHis	
Constant for key	0x0*
Timeout for key (ms)	100*
Call cycle	10*
CRC	Size optimized
Enable CRC Total	☐ *
File Selection	
Select security class	Class DDD

Figure 4-1 Security Component configuration

4.3 Configuration Parameters

Due to the OEM pre-configuration the following parameter may appear in the GENy configuration. Note that the default values may differ depending on the OEM.

Attribute Name	Default Values	Description
Constant for key	0x0	This is the key constant to adjust the authentication algorithm for a specific type of ECU.
Timeout for key	100 ms	This parameter is currently not used and is only provided for compatibility reasons to the HIS specification. The timeout handling of the authorization is managed within the diagnostic layer, not in the security component according to OEM specification and cannot be adjusted.
Call cycle	10	Specifies the call cycle to the function SecTask(). Commonly SecTask() is not used. In most cases this parameter is only for HIS-conformance.

Attribute Name	Default Values	Description
CRC	Size optimized	The CRC-module can run in two different modes: SIZE optimized and SPEED optimized. When 'SPEED optimized' is selected, a RAM table will be generated. If the checksum calculation shall also been used within the application, we strongly recommend to use "SIZE optimized".
Enable CRC Total	True	The security component always calculates a CRCtotal for a logical block when enabled. Thus, the application is able to verify the values within the application.

Table 4-1 GENy configuration parameter

4.4 Running the Generator

When you have finished your configuration selections, you need to run the generator to produce the files needed to compile the FBL. To generate the files, click on the lightning-bolt button on the toolbar, or select "Generate System" from the Configuration menu. The following table describes the contents of the generated files:

File Name	Description
Secm_cfg.h	The configuration file of the security module. For the most part the file defines switches in the form of SEC_ENABLE_<feature> or SEC_DISABLE_<feature> according to the settings in the dialog.
Secmpar.c, Secmpar.h	This file contains the secret key if security class C or the public key (on security class CCC) has been selected. The file is generated from the contents of the key given in the authenticity key file definition. You must compile and link these files with your security component.
v_cfg.h	Contains macro definitions to your hardware.
v_inc.h	Includes the generated headers, so that they may be obtained from a single source.
v_par.c, v_par.h	Contains information about your license. The files are not necessary for a successful compile/link procedure with the security component.

Table 4-2 Generated files

5 Preparation of the Application

Before a download can be made preparing the download file is a necessary step. For bootloaders with logical block tables, the address space used by one single file has to match information of a single block stored in the logical block table. It is possible to download multi region files, but the region count is limited by the Bootloader.

Each download file requires a signature or resp. a checksum file, which is created out of the file contents.

For Security class DDD, the checksum file contains 4 bytes, which reflects the CRC-32 checksum from the download file or 2 bytes in case of CRC-16.

To support you in the development process, the Bootloader product comes along with some tools to support you to create these checksum and signature files.

5.1 HexView

HexView is a software tool from Vector Informatik. It can be used to change and show the contents of a binary file, an Intel Hex file or a Motorola S-Record file. HexView has an extensive list of features. These features are important for the application:

> HexView can be used to fill the gaps of your application hex file and align the data to the memory segment size.

> HexView can be used to calculate the checksum or signature of your application.

Note
For detailed information about HexView please refer to the 'HexView Reference Manual' [3].

5.1.1 Fill and align the hex file with HexView

These are the command line arguments to fill and align a hex file:

>
```
hexview.exe demo.hex -S -e:error.txt -ad:0x08 -fa -xi -o
demo_out.hex
```
The hex file regions are aligned to 8 bytes addresses, the gaps are filled and the output is written into the file `demo_out.hex`

5.1.2 Create a checksum / signature file with HexView

These are the command line arguments for the signature and checksum calculation.

> Security Class DDD:
```
hexview.exe demo.hex -S -e:error.txt -cs9:demo.crc
```
The 4-byte CRC-32 value is directly written into the file demo.crc

6 Functional Description

This chapter describes the security modules, their variants and provides implementation samples for some common use-cases.

6.1 Verification Module

In the Bootloader case the security component is used to verify one or more sections of data. Depending on the security class, it consists of more or less complex mathematical routines that create a set of data with fixed length out of these sections of data. This fixed length data is called the signature or checksum.

It is specific to the mathematical routines that the signature changes drastically even though just a small portion or even a bit in the data stream has changed. It is also specific, that a key is necessary to create this signature.

6.1.1 Description of the verification process

As mentioned above, the signature is created on a set of data. This is usually a data stream that shall be downloaded and programmed into the memory of an ECU.

For this purpose, the flash memory is partitioned into logical blocks. Each logical block can store a program or other code that consists of one or more sections.

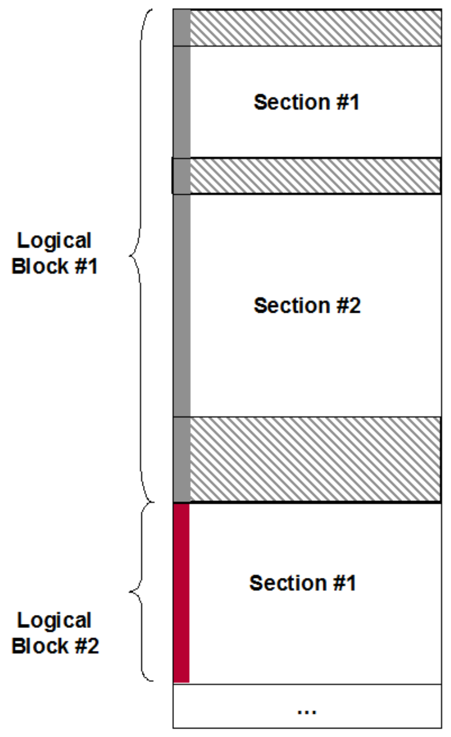


Figure 6-1 Logical Blocks

The CRC is created over the segments of a logical block. Thus, for the module of logical block #1, there are two segments available.

6.1.2 Usage

The following sample demonstrates how to use the verification module in monolithic use for verification on two sections and a logical block.

```
/* Function to read the data from memory */
SecM_SizeType ReadMemoryFct(SecM_AddrType address, vuint8 *buffer, SecM_SizeType length)
{
   memcpy(buffer, (void *)address, length);
}

/* Function is called frequently during the verification. */
/* At least in a shorter time distance than 1 ms.         */
void WatchdogFct( void )
{
   return 0u;
}

/* Verification parameters */
SecM_VerifyParamType verifyParam;

/* Signature to be verified */
vuint8 signature[128u];

/* List of memory segments */
FL_SegmentListType segmentList;

/* Setup section information of section #1 */
segmentList.segmentInfo[0].targetAddress      = FLASH_SECTION1_MEMORY_ADDRESS;
segmentList.segmentInfo[0].transferredAddress = FLASH_SECTION1_DOWNLOAD_ADDRESS;
segmentList.segmentInfo[0].length             = FLASH_SECTION1_LENGTH;

/* Setup section information of section #2 */
segmentList.segmentInfo[1].targetAddress      = FLASH_SECTION2_MEMORY_ADDRESS;
segmentList.segmentInfo[1].transferredAddress = FLASH_SECTION2_DOWNLOAD_ADDRESS;
segmentList.segmentInfo[1].length             = FLASH_SECTION2_LENGTH;

/* Number of memory segments */
segmentList.nrOfSegments                      = 2u;

/* Initialize verification structure */
/* Start address of full block */
verifyParam.blockStartAddress = FLASHREGION_LBT0_ADDRESS;
/* Length of full block */
```

```
verifyParam.blockLength        = FLASHREGION_LBT0_LENGTH;
/* Pass segment list */
verifyParam.segmentList        = &segmentList;
/* Function to read memory at given address */
verifyParam.readMemory         = ReadMemoryFct;
/* Cyclically called function */
verifyParam.wdTriggerFct       = WatchdogFct;
/* Pass signature */
verifyParam.verificationData   = (SecM_VerifyDataType)signature;
/* Use internal workspace */
verifyParam.workspace.size     = SEC_DEFAULT_WORKSPACE_SIZE;

/* Perform signature verification */
if ( (SECM_OK == SecM_InitVerification(V_NULL))
  && (SECM_VER_OK == SecM_Verification(&verifyParam))
  && (SECM_OK == SecM_DeinitVerification(V_NULL)) )
{
   /* Verification successful */
}
```

The example below shows the verification with the streaming use-case.

```
/* Verification parameters */
SecM_SignatureParamType sigParam;
/* Buffer for data to be verified */
vuint8 inputData[0x100u];
/* Control variable */
SecM_LengthType inputLength;
/* Read and hash input data */
SecM_AddrType inputAddress   = 0x10000u;
SecM_SizeType inputRemainder = 0x1000u;

/* Cyclically called function */
sigParam.wdTriggerFct = WatchdogFct;
/* Pass input buffer (not relevant for initialization) */
sigParam.sigSourceBuffer = inputData;

/* Initialize signature verification */
sigParam.sigState = SEC_HASH_INIT;
sigParam.sigByteCount = 0u;
```

```
if (SECM_VER_OK != SecM_VerifySignature(&sigParam))
{
   /* Initialization failed */
}

/* Proceed computation */
while (inputRemainder > 0u)
{
   /* Truncate input length */
   inputLength = sizeof(inputData);
   if (inputLength > inputRemainder)
   {
      inputLength = (SecM_LengthType)inputRemainder;
   }

   /* Read data into temporary buffer */
   sigParam.sigByteCount = (SecM_LengthType)ReadMemoryFct(inputAddress, inputData,
inputLength);

   if (SECM_VER_OK != SecM_VerifySignature(&sigParam))
   {
      /* Computation has failed */
      break;
   }

   /* Update control variables */
   inputRemainder -= inputLength;
   inputAddress   += inputLength;
}

/* Finalize hash calculation */
sigParam.sigState       = SEC_HASH_FINALIZE;
sigParam.sigByteCount   = 0u;

if (SECM_VER_OK == SecM_VerifySignature(&sigParam))
{
   /* Verify signature */
   sigParam.sigState          = SEC_SIG_VERIFY;
   /* Pass signature */
   sigParam.sigSourceBuffer   = signature;
   sigParam.sigByteCount      = sizeof(signature);
   if (SECM_VER_OK == SecM_VerifySignature(&sigParam))
   {
      /* Verification successful */
   }
}
```

6.2 CRC Module

The CRC is an error-detecting code which can be used to build a checksum over a defined segment. If this module was ordered a 16 or 32 Bit CRC implementation is included in the delivery. The SecM_ComputeCRC function can be called multiple times. The so called streaming approach can be used for the verification of a data stream. It can be necessary to verify the integrity of a logical block.

6.2.1 Variants

> CRC-16 (CCITT) or CRC-32 (IEEE 802.3)

> CRCTotal

> Speed/Size optimized

6.2.2 CRCTotal

To avoid storing the whole segment information with the resulting signature, the security module calculates a CRCTotal. This CRCTotal is generated internally by the security component during the verification process by the function SecM_Verification(). It is only necessary to pass the blockStartAddress and the blockLength of the logical block. The results will be placed into the CRCtotal field by the security module. If the CRCTotal option is activated in GENy it is always calculated, regardless of the security class.

Note
The CRCWritten is the CRC-value as the signature value calculated over the segments for the security class DDD.

The CRCwritten is not available for the Security Class C or CCC.

6.2.3 Speed / size optimization

In the GENy configuration the CRC module can be switched between two different modes: Speed or size optimized. When ‘SPEED optimized’ is chosen, a 1024 byte look-up table will be generated in case of CRC-32 or 256 byte for CRC-16.

Caution
If the checksum calculation shall also be used within the application, we strongly recommend to use ‘SIZE optimized’

6.2.4 Operation mode flags

These operation mode flags are valid and can be set by using the ‘crcState’.

> SEC_CRC_INIT

Sets the initial CRC value and initialize the look-up table.

> SEC_CRC_COMPUTE

Update the CRC calculation using the provided data.

> SEC_CRC_FINALIZE

Optional: Complete the CRC value

6.2.5 Usage

The following sample shows the usage of the CRC calculation. The usage of the module is similar to the stream verification.

The initialization sets up the environment:

```
/* CRC parameters */
SecM_CRCParamType crcParam;
/* Buffer for data to be verified */
vuint8            inputData[0x100u];

/* Cyclically called function */
crcParam.wdTriggerFct      = WatchdogFct;
/* Pass input buffer (not relevant for initialization) */
crcParam.crcSourceBuffer   = inputData;

/* Initialize signature verification */
crcParam.crcState          = SEC_CRC_INIT;
crcParam.crcByteCount      = 0u;

if (SECM_OK == SecM_ComputeCRC(&crcParam))
{
   ...
}
```

In the computation phase the processing will be done. Addresses and length information are typically not included in the CRC. The CRC Module doesn't have access to the memory in the function. Therefore a function to read from the memory is needed (in the example represented as 'ReadMemoryFct').

```
/* Control variable */
SecM_AddrType      inputAddress    = 0x10000u;
SecM_SizeType      inputRemainder = 0x1000u;
SecM_LengthType    inputLength;

/* Read input data and update checksum */
crcParam.crcState = SEC_HASH_COMPUTE;
while (inputRemainder > 0u)
```

```
{
   /* Truncate input length */
   inputLength = sizeof(inputData);
   if (inputLength > inputRemainder)
   {
      inputLength = (SecM_LengthType)inputRemainder;
   }

   /* Read data into temporary buffer */
   crcParam.crcByteCount = (SecM_LengthType)ReadMemoryFct(
      inputAddress, inputData, inputLength);

   if (SECM_VER_OK != SecM_ComputeCRC(&crcParam))
   {
      break;
   }

   /* Update control variables */
   inputRemainder -= inputLength;
   inputAddress   += inputLength;
}
```

During the finalization no additional data are required. The checksum is provided in the member currentCRC.

```
/* Checksum to be verified */
SecM_CRCType      checksum;

/* Finalize CRC calculation */
crcParam.crcState       = SECM_CRC_FINALIZE;
crcParam.crcByteCount   = 0u;

if (SECM_VER_OK == SecM_ComputeCRC(&crcParam))
{
   /* Compare checksum */
   if (crcParam.currentCRC == checksum)
   {
      /* Checksum comparison successful */
   }
}
```

6.3 Seed / Key

The Seed/Key interface provides the generation of a pseudo random seed, the generation of a key and a comparison function.

6.3.1 Usage

This demonstration generates a seed and computes a key. The usage of the module can vary between the OEMs. If the Security Module comes along with a Bootloader delivery the OEM related demo implementation can be found in the 'fbl_ap.c' file.

Seed generation:

```
SecM_WordType EntropySource( void )
{
   return <random value>;
}

/* Generate seed value */
SecM_SeedType seed;

/* Provide initial seed for pseudo random number generator */
seed.seedX = EntropySource();
seed.seedY = EntropySource();

/* Generate seed */
if (SECM_OK == SecM_GenerateSeed(&seed))
{
   /* Seed successfully generated in seed.seedX */
}
```

The provided key can be send to the tester. The tester has to generate a key and send it back to the ECU. The ECU has to generate with the seed value an own key and compare the received key from the tester with the generated one:

```
/* Check key from tester */
#if ( SECM_HIS_SECURITY_MODULE_VERSION < 0x010002u )
if (SECM_OK == SecM_CompareKey(key))
#else
if (SECM_OK == SecM_CompareKey(key, seed)
#endif
{
   /* Security access granted */
}
```

Note
The API varies depending on the version of the HIS specification. In case of HIS specification 1.0.1 or earlier the last seed is stored in an internal structure. Later versions do use an additional parameter for the seed. However the Security Module supports both variants.

6.4 Accessing the Security Module from within the Application

This chapter is only relevant if the security component comes along with the Bootloader.

There might be the necessity to use the features of the security module, located in the Bootloader, also within the application. This can be useful if functions of the Security Module shall be used in a consistent way, or ROM memory shall be saved. For this, a user-interface is available to call some routines of the security module from within the application. The file “fbl_def.h” contains the necessary definitions for the security module interface.

The usage of the definitions is equivalent to the API (chapter 7).

Definition	Description
ApplSecComputeKey	Definition for the key computation function.
ApplSecComputeCRC	Definition for the CRC computation function.
ApplSecVerifySignatureFct	Definition for the signature verification function.
ApplSecGenerateSeed	Definition for the seed generation function.

Table 6-1 Function access macros for the FBL header structure

7 API Description

7.1 Type Definitions

This chapter describes the types used by the interface functions of the security module.

7.1.1 General Types

Typedef	Elements	Basetype	Description
SecM_StatusType	Not particularly defined. Typically, returns SECM_OK or SECM_NOT_OK. See function description for details.	vuint8	Return value from any function of the security module
SecM_WordType	-	vuint32	Typically used by the seed/key functions for seed and key values.
SecM_KeyType	-	SecM_Word Type	The length of the key for seed/key calculation.
SecM_CRCType	-	SecM_Word Type	Used to store the CRC
SecM_LengthType	-	vuint32	Used to store length information for CRC calculation and decryption
SecM_AddrType	-	vuint32	Used to store address information for segments
SecM_SizeType	-	vuint32	Used to store length information of segments
SecM_VerifyInitType	-	void *	-
SecM_VerifyDeinitType	-	void *	-
SecM_VerifyDataType	-	vuint8 *	-
FL_WDTriggerFctType	typedef void (*)(void);		Pointer to a function that is frequently called from the security module on long-lasting mathematical functions.
FL_ReadMemoryFctType	typedef SecM_SizeType (*)(SecM_AddrType, vuint8 *, SecM_SizeType);		Function is used by SecM_Verification() to read data from memory.

Table 7-1 Type definitions

7.1.2 Structures used by Verification

Structure	Element	Type	Description
SecM_Verify	segmentList	FL_SegmentListType /	Address and length

Structure	Element	Type	Description
ParamType		FL_SegmentListType* [1]	information of download segments
	blockStartAddress	SecM_AddrType	Start address of logical block
	blockLength	SecM_SizeType	Length of logical block
	verificationData	SecM_VerifyDataType	Pointer to verification data (e.g. checksum and/or signature)
	crcTotal	SecM_CRCType	CRC total calculated over full logical block
	wdTriggerFct	FL_WDTriggerFctType	Pointer to watchdog trigger function
	workspace	SecM_WorkspaceType	Reference to workspace (extension of the HIS specification)
	key	SecM_VerifyKeyType	Pointer to verification key (extension of the HIS specification)

Table 7-2 SecM_VerifyParamType

Structure	Element	Type	Description
SecM_Signature ParamType	currentHash	SecM_SignatureType	Reference to current hash value (also used to pass workspace)
	currentDataLength	SecM_SizeType *	Pointer to the current length of the hashed data. Is only set when SEC_ENABLE_VERIFICATION_DATA_LENGTH is activated.
	sigState	SecM_StatusType	Signature state / operation to be executed
	sigSourceBuffer	SecM_VerifyDataType	Pointer to input or verification data
	sigByteCount	SecM_LengthType	Size of input or verification data
	wdTriggerFct	FL_WDTriggerFctType	Pointer to watchdog trigger function
	key	SecM_VerifyKeyType	Pointer to verification key (extension of the HIS specification)

[1] As an extension to the HIS specification the segment list can be used as a pointer. This depends on the OEM.

Table 7-3 SecM_SignatureParamType

7.1.3 Structures used by CRC

Structure	Element	Type	Description
SecM_CRCParamType	currentCRC	SecM_CRCType	Current CRC-value
	crcState	SecM_ByteType	Operation state to be carried out
	crcSourceBuffer	SecM_ConstRamDataType	Pointer to source data
	crcByteCount	SecM_LengthType	Number of bytes in source buffer
	wdTriggerFct	FL_WDTriggerFctType	Watchdog trigger function

Table 7-4 SecM_CRCParamType

7.1.4 Structures used by Seed / Key

Structure	Element	Type	Description
SecM_KeyType	data	SecM_KeyBaseType *	Pointer to actual key data
	context	SecM_VoidPtrType	Additional context information, e.g. workspace

Table 7-5 SecM_KeyType

7.2 Common

7.2.1 SecM_InitPowerOn

Prototype	
`SecM_StatusType` **`SecM_InitPowerOn`** `(SecM_InitType initParam)`	
Parameter	
SecM_InitType initParam	Initialization parameters (unused - reserved for future use)
Return Code	
SecM_StatusType	SECM_OK if initialization successful SECM_NOT_OK otherwise
Functional Description	
Initialize security module	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
none	

Table 7-6 SecM_InitPowerOn

7.2.2 SecM_Task

Prototype	
`void SecM_Task (void)`	
Parameter	
void	none
Return Code	
void	none
Functional Description	
Cyclic task of security module	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
none	

Table 7-7 SecM_Task

7.3 CRC

7.3.1 SecM_ComputeCRC

Prototype	
`SecM_StatusType SecM_ComputeCRC (V_MEMRAM1 SecM_CRCParamType V_MEMRAM2 V_MEMRAM3 *crcParam)`	
Parameter	
SecM_CRCParamType crcParam	Pointer to parameter structure
Return Code	
SecM_StatusType	SECM_OK if operation was successful SECM_NOT_OK otherwise
Functional Description	
Function that manages the state of CRC computation	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
none	

Table 7-8 SecM_ComputeCRC

7.4 Seed / Key Common

7.4.1 SecM_GenerateSeed

Prototype	
`SecM_StatusType` **`SecM_GenerateSeed`** `(V_MEMRAM1 SecM_SeedType V_MEMRAM2 V_MEMRAM3 *seed)`	
Parameter	
SecM_SeedType seed	Pointer to seed, where the random values shall be stored
Return Code	
SecM_StatusType	SECM_OK
Functional Description	
Uses a pseudo random number generator and an initial seed to generate a seed value. In case of HIS specification 1.0.1 or earlier the last seed is stored in an internal structure.	
Particularities and Limitations	
none	
Pre-Conditions	
The values of seedX and seedY (default API) or seed[0] (extended API) should have a random start value.	
Call Context	
none	

Table 7-9 SecM_GenerateSeed

7.4.2 SecM_CompareKey (HIS Specification < 1.0.1[2])

Prototype	
`SecM_StatusType` **`SecM_CompareKey`** `(SecM_KeyType key)`	
Parameter	
SecM_KeyType key	Key for the authorisation
Return Code	
SecM_StatusType	SECM_OK if key calculation and comparison was successful SECM_FAILED if key calculation failed or key mismatch
Functional Description	
Runs the key calculation and compares the calculated key against the received key	
Particularities and Limitations	
none	
Pre-Conditions	
Function SecM_GenerateSeed must have been called before.	
Call Context	

[2] Depending on the OEM

none

Table 7-10 SecM_CompareKey

7.4.3 SecM_CompareKey (HIS Specification >= 1.0.1)

Prototype	
`SecM_StatusType` **`SecM_CompareKey`** `(SecM_KeyType key, SecM_SeedType lastSeed)`	
Parameter	
SecM_KeyType key	Key for the authorization contains additional parameters (workspace, secret key) in case of extended API
SecM_SeedType lastSeed	Start value (seed) for the authorisation
Return Code	
SecM_StatusType	none
Functional Description	
Runs the key calculation and compares the calculated key against the received key	
Particularities and Limitations	
none	
Pre-Conditions	
Function SecM_GenerateSeed must have been called before.	
Call Context	
none	

Table 7-11 SecM_CompareKey

7.5 Seed / Key default API

7.5.1 SecM_ComputeKey

Prototype	
`SecM_StatusType` **`SecM_ComputeKey`** `(SecM_SeedType inputSeed, SecM_ConstType constant, V_MEMRAM1 SecM_KeyStorageType V_MEMRAM2 V_MEMRAM3 *computedKey)`	
Parameter	
SecM_SeedType inputSeed	The random seed the key calculation shall be based on
SecM_ConstType constant	A fixed constant used by the key calculation algorithm contains additional parameters (workspace, secret key) in case of extended API
SecM_KeyStorageType computedKey	Pointer to resulting key data as a formula of key = f(seed,k)
Return Code	
SecM_StatusType	SECM_OK if calculation was successful SECM_FAILED if calculation has failed (e.g. wrong parameters)

Functional Description
Calculates a key value based on the provided seed and constant value
Particularities and Limitations
none
Pre-Conditions
Function SecM_GenerateSeed must have been called at least once.
Call Context
none

Table 7-12 SecM_ComputeKey

7.6 Seed / Key extended API

7.6.1 SecM_ComputeKey

Prototype	
`SecM_StatusType` **`SecM_ComputeKey`** `(SecM_KeyType inputKey, SecM_ConstType constant, V_MEMRAM1 SecM_SeedType V_MEMRAM2 V_MEMRAM3 *computedSeed)`	
Parameter	
SecM_KeyType inputKey	The received key on which calculation shall be based
SecM_ConstType constant	A fixed constant used by the key calculation algorithm contains additional parameters (workspace, secret key) in case of extended API
SecM_SeedType computedSeed	Pointer to resulting seed as a formula of seed = f(key,k)
Return Code	
SecM_StatusType	SECM_OK if calculation was successful SECM_FAILED if calculation has failed (e.g. wrong parameters)
Functional Description	
API is a deviation from the HIS specification. Modified to support asymmetric key calculation where only the tester is able to generate the key from the original seed value, e.g. using a private key.	
Particularities and Limitations	
none	
Pre-Conditions	
Function SecM_GenerateSeed must have been called at least once.	
Call Context	
none	

Table 7-13 SecM_ComputeKey

7.7 Verification

7.7.1 SecM_InitVerification

Prototype	
`SecM_StatusType` **`SecM_InitVerification`** `(SecM_VerifyInitType init)`	
Parameter	
SecM_VerifyInitType init	Dummy pointer (currently not used)
Return Code	
SecM_StatusType	SECM_OK if initalization successful SECM_NOT_OK if error occured during initalization
Functional Description	
Initializes the verification	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
none	

Table 7-14 SecM_InitVerification

7.7.2 SecM_DeinitVerification

Prototype	
`SecM_StatusType` **`SecM_DeinitVerification`** `(SecM_VerifyDeinitType deinit)`	
Parameter	
SecM_VerifyDeinitType deinit	Dummy pointer (currently not used)
Return Code	
SecM_StatusType	SECM_OK if deinitalization successful SECM_NOT_OK if error occured during deinitalization
Functional Description	
Deinitializes the verification	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
none	

Table 7-15 SecM_DeinitVerification

7.7.3 SecM_Verification

Prototype	
`SecM_StatusType` **`SecM_Verification`** `(V_MEMRAM1 SecM_VerifyParamType V_MEMRAM2 V_MEMRAM3 *pVerifyParam)`	
Parameter	
SecM_VerifyParamType pVerifyParam	Pointer to parameter structure for signature verification crcTotal.
Return Code	
SecM_StatusType	SECM_VER_OK if signature verification successful SECM_VER_ERROR if error occured during verification SECM_VER_CRC if CRC verification failed SECM_VER_SIG if HMAC/Signature verification failed
Functional Description	
Calculates the checksum or signature of all segments present in the verification parameter function passed in readMemory is used to access memory. Afterwards the calculated value is used to verify the checksum/signature provided in verificationData. If configured a CRC over the complete block, defined by blockStartAddress and blockLength, including all readable gaps between the actual segments is calculated too and returned in parameter	
Particularities and Limitations	
none	
Pre-Conditions	
Call Context	
none	

Table 7-16 SecM_Verification

8 Contact

Visit our website for more information on

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

www.vector.com