SHE Key Update Protocol

Technical Reference

How to Persist SHE Key in MICROSAR and Functional Flow

Version 1.00.00

Authors	Subrahmanyam Namdikam
Status	Released

Document Information

History

Author	Date	Version	Remarks
Subrahmanyam Namdikam	[2020-04-01]	1.0.0	First Version

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_CryptoDriver.pdf	AR 4.4.0
[2]	SHE Functional Specification	2009-04-01 SHE Functional Specification v1.1 (rev439).pdf	
[3]	Vector	TechnicalReference_Crypto_30_LibCv.pdf	see delivery

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

Contents

1 Introduction About SHE 6

2 SHE Key Update Protocol 7

2.1 Crypto Principles of SHE Key Storage: 7

2.2 Different types of Keys which will be stored in the SHE 7

3 Key Update Flow 8

3.1 Kauth, Knew, CID, UID, FID to the Generator 9

3.2 FID: Protection Flags 9

3.3 Generation of M1 10

3.4 Generation of M2 10

3.5 Generation of M3 11

3.6 Generation of M4 11

3.7 Generation of M5 12

4 SHE Key Update Configuration 13

4.1 Master_Ecu_Key, Secret_Key, Boot_Mac, Boot_Mac_key 13

4.2 SHE CID, FID& She Info 14

4.3 Details of the Parameters of Key Element Configuration 16

4.4 SHE Page Update Constants 16

4.5 SHE Key_<n> Configuration 17

4.5.1 SHE_Key (Which is the actual SHE key) 19

4.5.2 SHE Key Proof 19

4.5.3 SHE Counter 20

5 Flow Chart 21

6 Code Flow 22

6.1 Call Stack from CSM (Csm_KeyElementset) 23

6.2 Crypto_30_LibCv_SheKeyUpdate 24

6.3 Crypto_30_LibCv_SheKeyUpdateCheckM1Ids 25

6.4 Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract 26

6.5 Crypto_30_LibCv_SheKeyUpdateVerifyM3 27

6.6 Crypto_30_LibCv_SheKeyUpdateExtractKey 27

6.7 Csm_KeyElementSetValid Handling 28

7 Appendix 29

7.1 Appendix A 29

7.1.1 How to Get Stored Key Details 29

7.1.2 Crypto NVM Block Configuration 30

7.2 Appendix B 31
7.2.1 Call Stack of MAC Generate & Verify 31
7.2.2 Call Stack of Encrypt, Decrypt & Random Generate 32

8 Glossary and Abbreviations 34
8.1 Abbreviations 34

9 Contact 35

Illustrations

Figure 1-1 Simplified SHE Block Diagram 6
Figure 3-1 One Typical Example, How the SHE Key Will be Updated by the Tester 8
Figure 3-2 Structure of M1 10
Figure 3-3 Structure of M2 10
Figure 3-4 Structure of M3 11
Figure 3-5 Structure of M4 11
Figure 3-6 Structure of M5 12
Figure 4-1 Master, Secret, Boot_Mac Keys & Boot_Mac Configuration 13
Figure 4-2 Example Configuration of Master Key Under Crypto Keys 13
Figure 4-3 Structure of Crypto_30_Libcv_SheKeys 14
Figure 4-4 Example Configuration of CID, FID 14
Figure 4-5 Key Elements of UID, Boot Protection, Debugger Protection Mapped to the Key 15
Figure 4-6 UID Key Element Configuration 15
Figure 4-7 Configuration for Constants 16
Figure 4-8 Normal SHE Key Configuration 17
Figure 4-9 Normal SHE Key to Key Element Mapping 18
Figure 4-10 Actual SHE Key Element Configuration 19
Figure 4-11 SHE Key Proof (Key Element) Configuration 19
Figure 4-12 SHE Counter (Key Element) Configuration 20
Figure 5-1 Simple Flow Chart to Persist the SHE Key 21
Figure 6-1 Prototype of Csm_KeyElementSet, Csm_KeySetValid 23
Figure 6-2 Call Stack of the Csm_KeyElementSet Function 23
Figure 6-3 Structure of CryIf_CryptoFunctions 24
Figure 6-4 Prototype of Crypto_30_Libcv_KeyElementSet 24
Figure 6-5 Crypto_30_LibCv_SheKeyUpdate Function 25
Figure 6-6 Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract Function 26
Figure 6-7 Possible key Validity Functions in the Crypto Driver 28
Figure 7-1 Crypto_30_LibCv_KeyElements Structure Format (Crypto SW) 29
Figure 7-2 Crypto_30_LibCv_KeyElements Structure Format (vHsm) 29
Figure 7-3 Calculation to Extract Valid Index, Start Index , End Index (vHsm) 30
Figure 7-4 Crypto NVM Block Configuration 30
Figure 7-5 Call stack of MAC Generate 31
Figure 7-6 Call Stack of MAC Verify 32
Figure 7-7 Call Stack of 27 Service Decrypt Key (AES Decrypt) 32
Figure 7-8 Call Stack of 27 Service Encrypt Seed (AES Encrypt) 32
Figure 7-9 Call Stack of 27 Service Random Number Generation 33

Tables

Table 3-1 Table Indicates Which Flags are Valid to Which Keys 10
Table 4-1 Provides Fixed SHE IDs in SHE Page Zero 14

1 Introduction About SHE

The Secure Hardware Extension (SHE) is an on-chip extension to any given microcontroller. It is intended to move the control over cryptographic keys from the software domain into the hardware domain and therefore protect those keys from software attacks.

As shown in the figure below, SHE provides secured storage to store keys. It also supports basic symmetric primitive (AES).

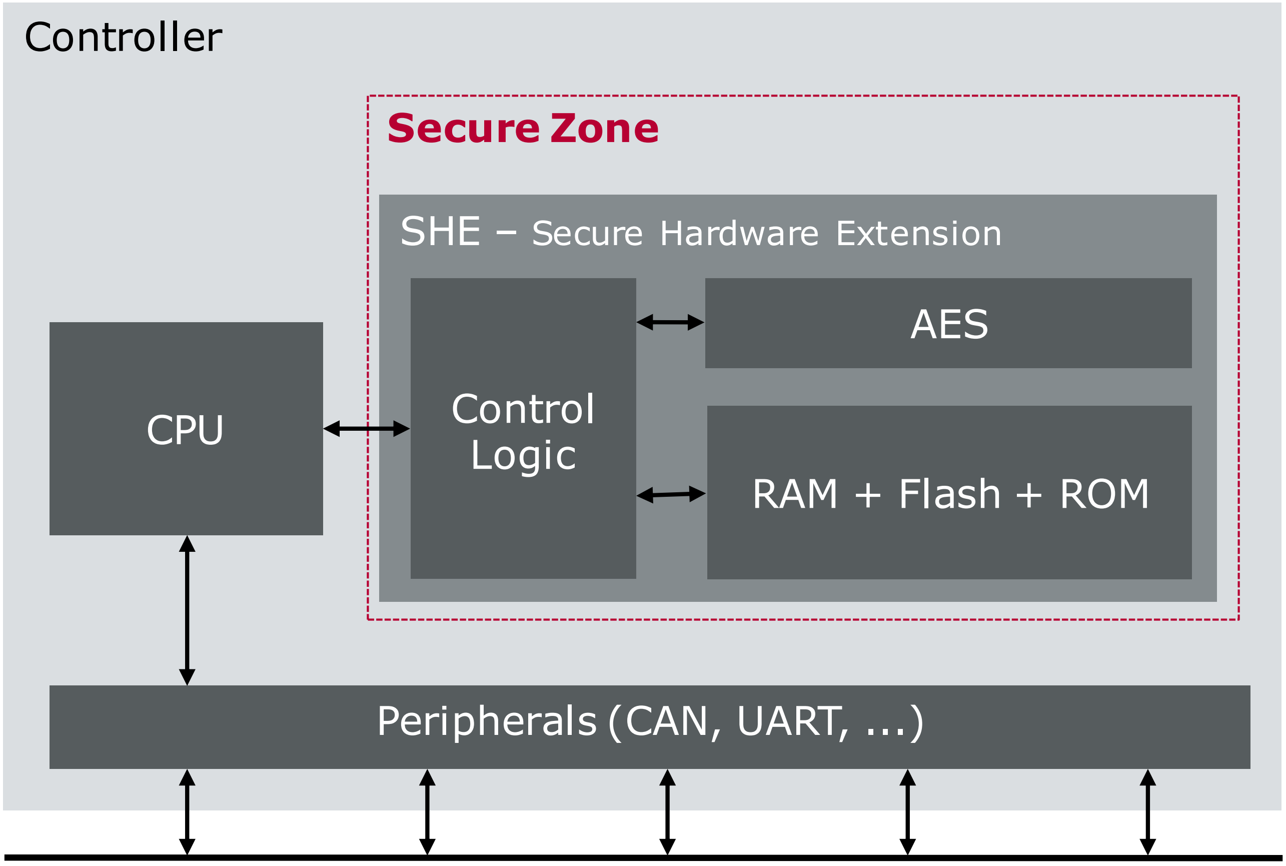

Figure 1-1 Simplified SHE Block Diagram

2 SHE Key Update Protocol

SHE provides several memory slots to store keys and provides a feasibility to update these stored keys with additional checks through flags & counter values. See section 3.1 and 3.2.

2.1 Crypto Principles of SHE Key Storage:

While updating the Keys it follows basic Crypto Principles

> **Authenticity**: While Updating the Key Slot, Sender and Receiver use the Authentication key (Shared Symmetric Key) to validate the transferred information. Receiver issues the Verification message to the Sender to prove the successful update.

  Integrity: To avoid Replay attacks, Counter values are also stored along with the key persisting in the SHE Hardware. These counter values are incremented for every successful update of SHE key.

> **Confidentiality:** Where SHE key, Counter value, Flags and Verification Result be encrypted before sharing.

2.2 Different types of Keys which will be stored in the SHE

> **SECRET_KEY:** Which will be stored in the ROM. This will be persisted during chip fabrication by the semiconductor manufacturer. This key will be used to Import / Export Keys.

> **MASTER_ECU_KEY *:** Which will be stored in the Non-Volatile Memory slot. This key will be persisted by the Owner of component (ECU Owner or OEM) using SHE protocol. Master Key will be used to reset or Change the SHE keys. Master key can be used as Authentication key while updating the other normal SHE keys (Key_<n>). MASTER key can be re written with the knowledge of other Master key

> **KEY_<n> *:** These are SHE keys which will be stored in the Non-Volatile Memory Slot. Here n is an arbitrary number which can be vary from 0 to 20. These keys generally used for Encryption/Decryption, MAC generation /Verification. Key_<n> can be persisted with the knowledge of MASTER key or the Current Key_<n>. In the Configuration Write access is **WA_ENCRYPTED**.

> **BOOT_MAC_KEY *:** The BOOT_MAC_KEY is used by the secure booting mechanism to verify the authenticity of the software. The BOOT_MAC_KEY may also be used to verify a MAC. The BOOT_MAC_KEY can be written with the knowledge of the MASTER_ECU_KEY or BOOT_MAC_KEY

> **BOOT_MAC *:** The BOOT_MAC is used to store the MAC of the Bootloader of the secure booting mechanism. The BOOT_MAC can be written with the knowledge of the MASTER_ECU_KEY or BOOT_MAC_KEY.

> **RAM_KEY:** This is plain text key and this can be written with the knowledge of the KEY_<n> or in plain text. For the Plain Text keys, the configuration parameters of write access as WA_ALLOWED and read access as RA_ALLOWED be configured.

*: These keys must be empty after production.

3 Key Update Flow

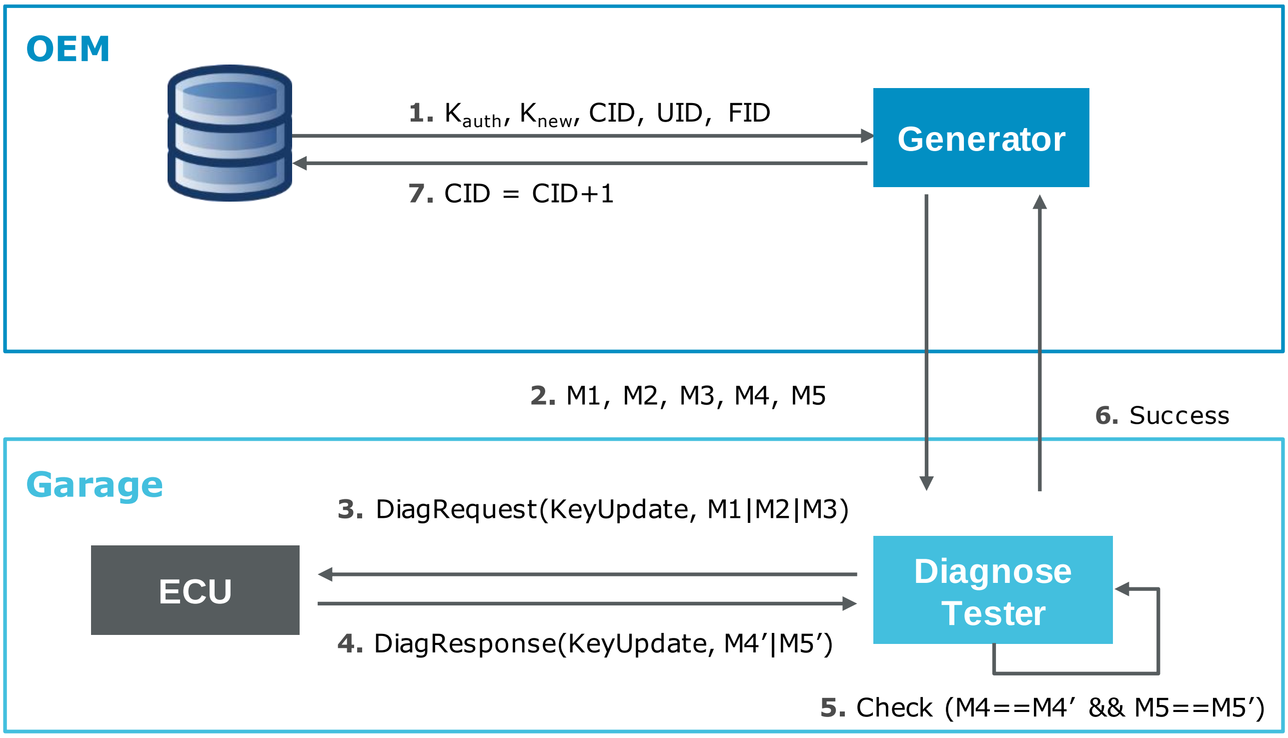

Figure 3-1 One Typical Example, How the SHE Key Will be Updated by the Tester

1. **Step 1**: OEM or User will provide the Kauth, Knew, CID, UID, FID to the Generator
2. **Step 2**: Generator Generates M1, M2, M3, M4 & M5 (Refer sections from 3.3 to 3.7) values and provides to the Tester
3. **Step 3**: Tester sends the M1, M2, M3 values, which holds the actual key to be persisted, as well as further Authentication key details.

   ECU decrypts the data by using Authentication Key and other Generated keys (K1, K2 Refer section 3.4 & 3.5). Actual SHE key will extracted from the decrypted data.

   ECU verifies the M3 value and SHE key be persisted after M3 verification successful.
4. **Step 4**: If user configures the proof, ECU calculates M4' & M5' and sends to the Tester.
5. **Step 5**: Tester receives the M4' & M5' and compares with M4 & M5.
6. **Step 6**: If the M4 & M5 comparison (Step5) successful, Tester sends the success result to the Generator
7. **Step 7**: Generator increments the CID (Counter Identifier value) and sends to the OEM, where this incremented Counter value is being used for the next key persisting.

Reference
Please refer chapters below for more details of the M1, M2, M3, M4 & M5, UID, CID, FID, Kauth, Knew.

3.1 Kauth, Knew, CID, UID, FID to the Generator

OEM has to provide the Kauth, Knew, CID, UID, FID to the Generator

- **UID:** Unique Identification Identifier, Ex : ECU Unique ID. The size of the UID is 120 bits. The UID has to be inserted during chip fabrication by the semiconductor manufacturer. UID should not be set as Zeros at least one bit should be set. UID Zeros are allowed only when Wild Card Flag has been disabled.
- **Kauth**: Authentication Key (Either this would be a Master key or old key)
- **Knew:** New key value which has to be persisted (KEY_{ID})
- **CID**: Counter Value, which is bigger than the Counter value which has been stored internally (this Counter value be incremented by one for every successful key persisting and this will be used for next key persisting).

3.2 FID: Protection Flags

FID = WRITE_PROTECTION|BOOT_PROTECTION|DEBUGGER_PROTECTION|KEY_USAGE|WILDCARD

- `WRITE_PROTECTION`: if this flag is set "1", key slot can't be overwritten.
- `BOOT_PROTECTION`: If this flag is set to "1", key can't be used if the Secured boot process has not finished or failed.
- `DEBUGGER_PROTECTION`: If this flag is to "1", key can't be used if the debugger is attached on device.
- `KEY_USAGE`: Determines if a key can be used for en/decryption or MAC. Flag set means that it`ll be used for MAC generation/verification otherwise it can be used for en/decryption.
- `WILDCARD`: UID can be replaced with Wild card value (Zeros) when this flag is disabled.
- `CMAC USAGE`: This flag is only checked is KEY USAGE is "1" otherwise it shall be "0", If the flag is set the key can only be used for CMAC verification otherwise used for Generation

FID / SHE Key	WRITE PROTECTION	BOOT PROTECTION	DEBUGGER PROTECTION	KEY USAGE	WILDCARD	CMAC USAGE	Counter
SECRET_KEY		■*	■*				
MASTER_ECU_KEY	■	■	■		■		■
BOOT_MAC_KEY	■		■		■		■
BOOT_MAC	■		■		■		■
KEY_{n}	■	■	■	■	■	■	■
RAM_KEY_{Page}							
* use flags from master key							

Table 3-1 Table Indicates Which Flags are Valid to Which Keys

3.3 Generation of M1

In the figure below, SHE ID (4bits) provides the respective key slot number from the ECU, where the SHE key has to be persisted.

AuthID (4bits) provides the Authentication Key Slot number, where respective key be used for Authentication.

If AuthID slot is same as ID slot: Previous persisted SHE key in the same ID slot be used as Authentication Key. AuthID slot is “1” to use Master key as Authentication key.

UID (120 bits)	**SHE ID** (4 bits)	**AuthID** (4 bits)

Figure 3-2 Structure of M1

3.4 Generation of M2

M2 (256 bits)	$ENC_{CBC, K1, IV=0}$			
	CID:Counter (28 bits)	**FID (Flag)** WP BP DP KI WC CU (6 bits)	**„0…0“** (94 bits)	**KEY_{ID}** (128 bits)

Figure 3-3 Structure of M2

> $M2 = ENC_{CBC,\,K1,IV=0}(CID|FID|\ \text{"0…0"}_{95}\ |KID)$

KEY_{ID}: New Key value which has to be persisted.

The Concatenated string of CID, FID, Zeros and KEYID will be given as input string for the AES Algorithm, which follows CBC method. Input Vector is Zeros and another key is K1.

K1 = AESMP(K_auth ┤|KEY_UPDATE_ENC_C)

K_auth: Value of the authentification key used for the Key Update.

KEY_UPDATE_ENC_C: constant value.

AESMP is AES128 - Miyaguchi-Preneel construction.

3.5 Generation of M3

M3 is verification Message and is calculated as $CMAC_{K2}$ (M1|M2)

M3 (128 bits)	$\mathbf{CMAC_{K2}}$ (M1|M2)

Figure 3-4 Structure of M3

$\boldsymbol{M1} = UID \mid ID \mid AuthID$

$\boldsymbol{M2} = ENC\_\ (CBC, K1, IV{=}0)\ (CID^{\prime} ┤|FID^{\prime} \mid$ 〖"\"0…0\""〗 $\_\ (95)\ |KID^{\prime})$

$\boldsymbol{K2} = AESMP\ (K\_auth\ |KEY\_UPDATE\_MAC\_C)$

$\boldsymbol{K\_auth}$: Value of the authentication key used for the Key Update.

$\boldsymbol{KEY\_UPDATE\_MAC\_C}$: constant value.

3.6 Generation of M4

M4 is being used for verification

M4 (256 bits)	UID (120 bits)	ID (4 bits)	Auth ID (4 bits)	$ENC_{ECB,K3}$		
				CID:Counter (28 bits)	„1" (1 bit)	„0…0" (99 bits)

Figure 3-5 Structure of M4

M4 Is the Verification message, which contains the M1 information and additionally Encrypted Counter Value.

M4 = M1 | M4* M4 * = $ENCECB,K3(CID)$

$\boldsymbol{K3} = AESMP\ (K\_new\ |KEY\_UPDATE\_ENC\_C)$, where $KEY\_UPDATE\_ENC\_C$ is Constant

$\boldsymbol{K\_new}$: Value of the new key used for the Key Update.

3.7 Generation of M5

M5 (128 bits)	**$CMAC_{K4}$** (M4)

Figure 3-6 Structure of M5

K4 = AESMP(Knew | KEY_UPDATE_MAC_C) , Where KEY_UPDATE_MAC_C is Constant

Knew is the Value of the new key used for the Key Update.

M5 is generated by calculating a CMAC over the message M4 with a key K4 derived from the updated memory slot ID and KEY_UPDATE_MAC_C.

If the returned M4 & M5 matches with the Generated M4 & M5 values, the Key Update Protocol was completed successfully.

4 SHE Key Update Configuration

4.1 Master_Ecu_Key, Secret_Key, Boot_Mac, Boot_Mac_key

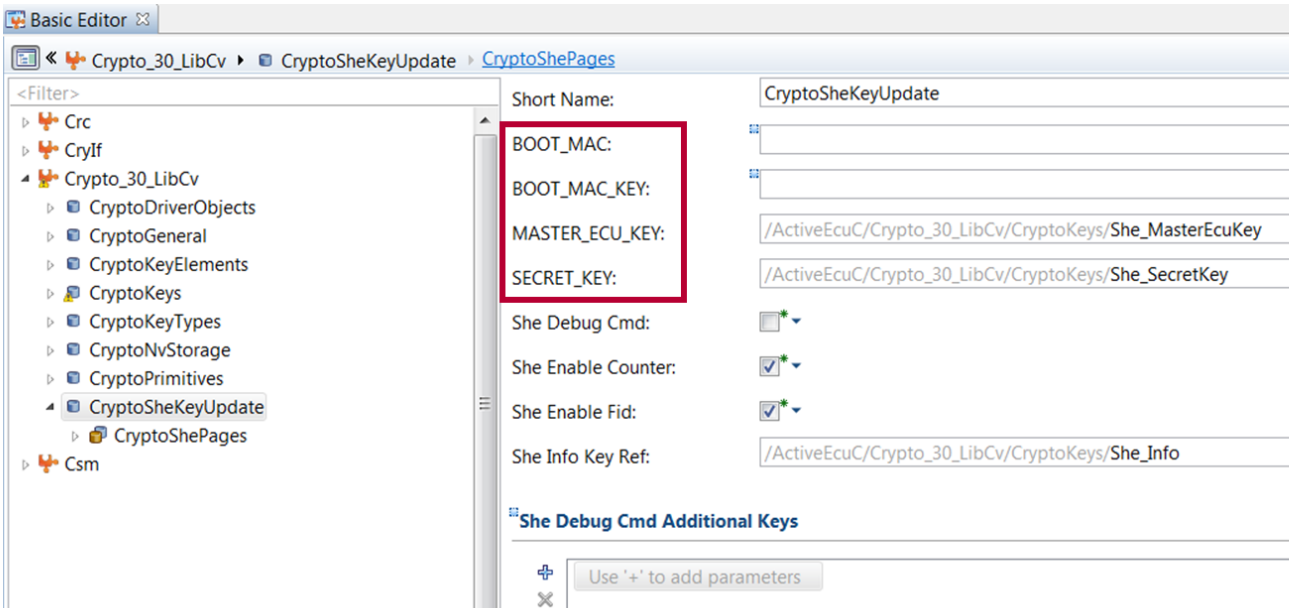

Figure 4-1 Master, Secret, Boot_Mac Keys & Boot_Mac Configuration

Refer 2.2 for details of these keys. Above configuration represents at ECU side, where respective key elements should be mapped through the Crypto Keys.

Example
In the Figure 4-1 MASTER_ECU_Key has been mapped with the She_Master_Key, which is from the Crypto Key (as shown in below Figure 4-2).

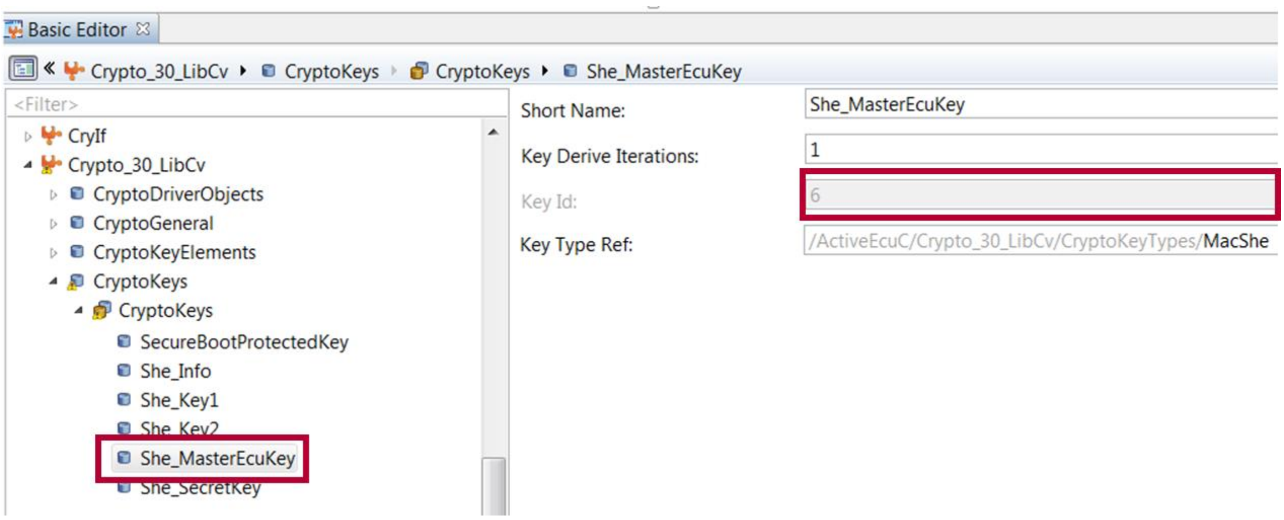

Figure 4-2 Example Configuration of Master Key Under Crypto Keys

Key ID of the Master key is 6, which has been reflected in the below structure `Crypto_30_Libcv_SheKeys` as shown in the Figure 4-2.

```
CONST(Crypto_30_LibCv_SheKeysType, CRYPTO_30_LIBCV_CONST) Crypto_30_LibCv_SheKeys[56] = {  /* PRQA S 1514, 1533 */
    /* Index    KeyElementsCounterIdx  KeyElementsKeyIdx  KeyIdx  SheId  ShePageIdx        Referable Keys */
  { /*     0 */                  21u,               19u,     7u,    0u,         0u },  /* [/ActiveEcuC/Crypto_30_LibCv/CryptoKeys/She_SecretKey] */
  { /*     1 */                  18u,               16u,     6u,    1u,         0u },  /* [/ActiveEcuC/Crypto_30_LibCv/CryptoKeys/She_MasterEcuKey] */
  { /*     2 */                   8u,                6u,     3u,    4u,         0u },  /* [/ActiveEcuC/Crypto_30_LibCv/CryptoKeys/She_Key1] */
```

Figure 4-3 Structure of Crypto_30_Libcv_SheKeys

Note
For the Master Key the SHE page is fixed to Zero and SHE ID is also fixed to 1.

Hence during M1 calculation AUTHID slot and SHE ID slot should be the same, which should be "1" (Refer section 3.3).

The table below shows that the keys are existing in the SHE page Zero and respective SHE IDs are also fixed.

SHE Key	SHE ID	SHE Page
Security Key	0	0
Master ECU Key	1	0
Boot MAC Key	2	0
Boot MAC	3	0

Table 4-1 Provides Fixed SHE IDs in SHE Page Zero

4.2 SHE CID, FID& She Info

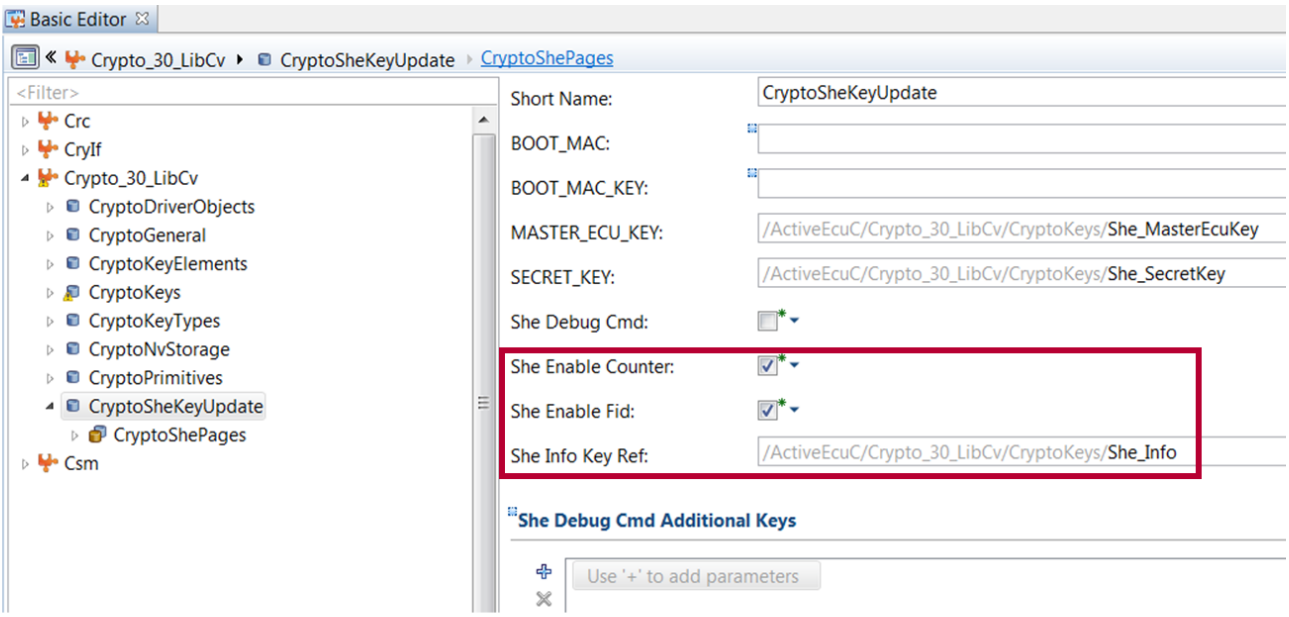

Figure 4-4 Example Configuration of CID, FID

> **SHE Enable Counter** (CID): This Parameter should be enabled, if user want to consider the counter to be validated during persisting of SHE Keys. Reg the CID refer section 3.1 & 3.4.

> **SHE Enable FID**: Refer section 3.2. If FID check is disabled all protection flag values will be interpreted by the Software as Zero. (key usage will be used only for En/Decryption and CMAC usage bit not being used)

> **SHE Info Key Ref**: Reference to the key which holds the UID Key, Boot Protection & Debugger Protection Control Key Elements. Where Boot & Debugger Protection Key elements are needed only, when **SHE Enable FID** has been enabled and its size is limited to 1 byte. Boot & Debugger protections key elements are normal plain text keys and hence Read Access & Write Access be configured as ALLOWED.

> Respective Key Ids of Boot Protection & Debugger protection are 3056 & 3057.

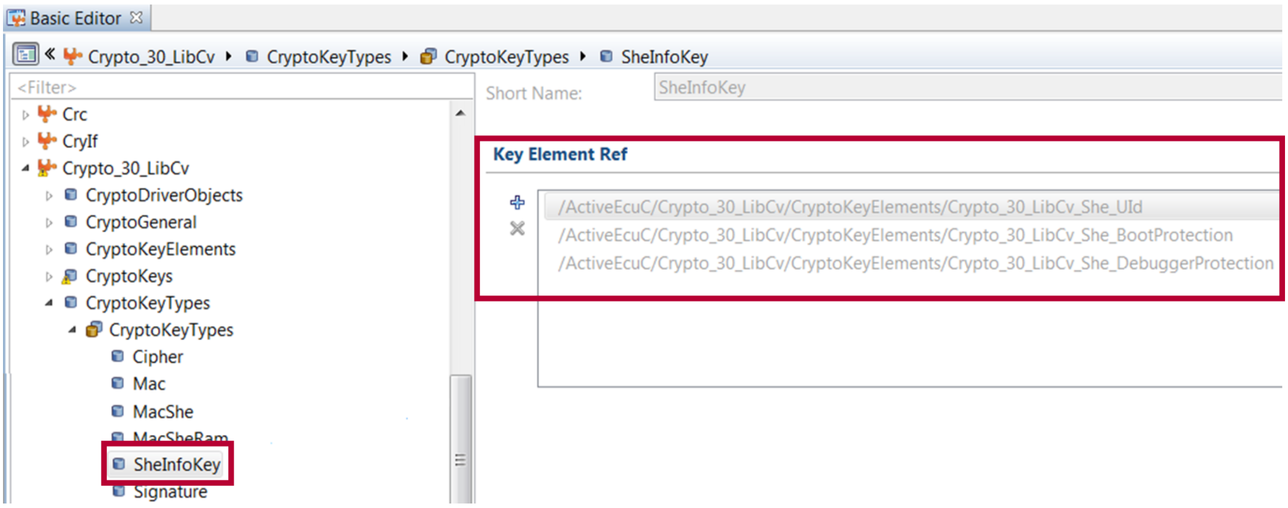

Figure 4-5 Key Elements of UID, Boot Protection, Debugger Protection Mapped to the Key

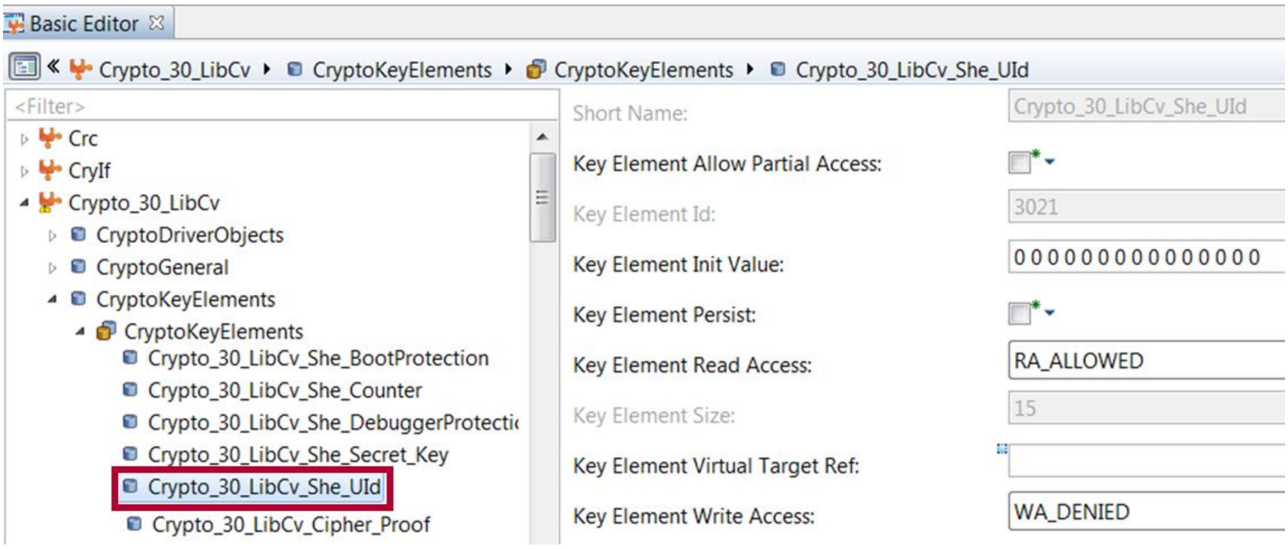

Figure 4-6 UID Key Element Configuration

Reg the UID refer section 3.1.

The screenshot above indicates the key element configuration for the UID (it is just a sample configuration).

UID is used as ECU ID, which is a unique value for each hardware, This UID value should be the same at both Sender & Receiver side. This can be used to generate encrypted key information (M1|M2|M3) which is only valid for a specific ECU or FID wildcard support is used.

4.3 Details of the Parameters of Key Element Configuration

> **Key Element ID**: Here it is fixed to 3021 from the pre config file

> **Key Element Init Value**: Value which will be used to fill the element during initialization, when the element values is not been persisted or issue during reading of persisted values occurs.

> **Key Element Persist**: By enabling this, key element can be written in to NVM.

> **Key Element Size**: According to SHE protocol the length of the UID is 120 bits (15 Bytes)

> **Key Element Read Access**: This should be configured as RA _ALLOWED, which means reading the key is allowed as plain text key.

> **Key Element Write Access**: Key element can’t be written externally, Ex : by using the KeyElementSet function .

4.4 SHE Page Update Constants

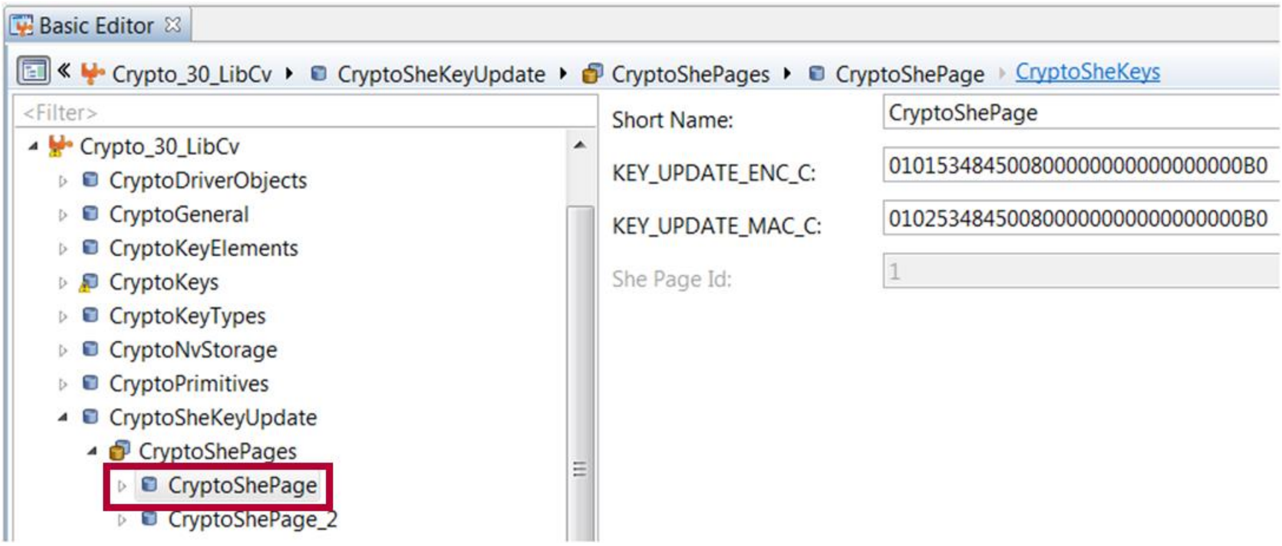

Figure 4-7 Configuration for Constants

Refer section from 3.4 to 3.7 for the Key Update Constant usage, where these constants are being used to Generate Back ground keys K1, K2, K3 & K4. Same Constants be used at both Sender & receiver side. These constants are configured at each Page level.

Note
Page 0 related keys (Master Key, Secret key, Boot MAC key & Boot Mac) uses Constants which are configured to SHE page ID 1

SHE page Constants which have been used in M2, M3, M4 & M5, are selected through the structures below in the software.

```
CONST (Crypto_30_LibCv_ShePageType, CRYPTO_30_LIBCV_CONST)
Crypto_30_LibCv_ShePage[x],
```

which provides the Start & End Indexes for both ENC, MAC constants.

Based on the Index from the above structure, the constants values are extracted from the array `Crypto_30_LibCv_SheConstants [x].`

4.5 SHE Key_<n> Configuration

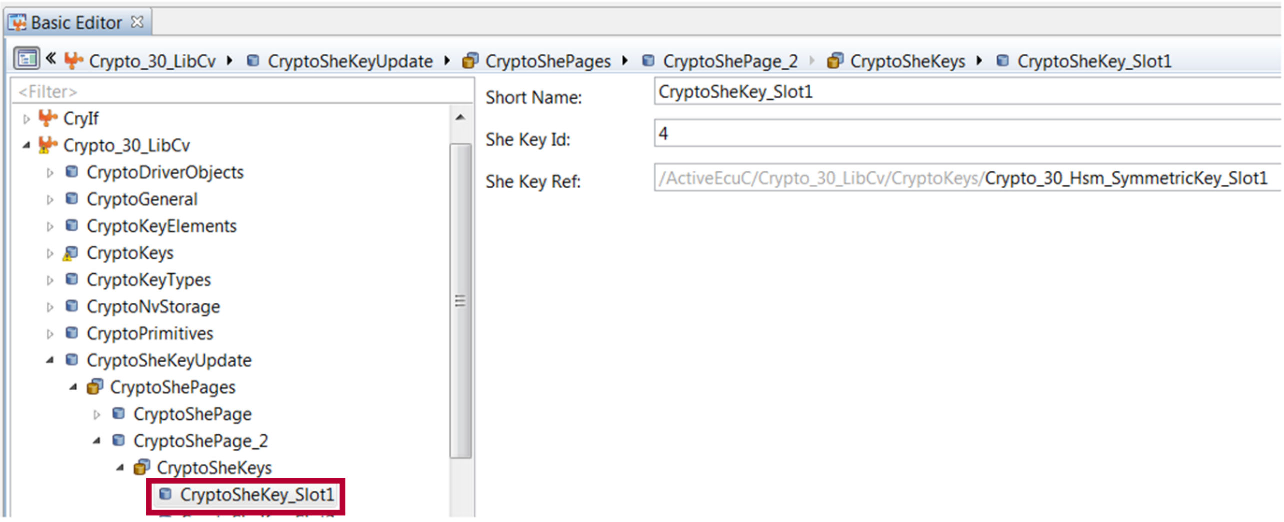

Figure 4-8 Normal SHE Key Configuration

Note
All Normal SHE keys IDs start from 4.

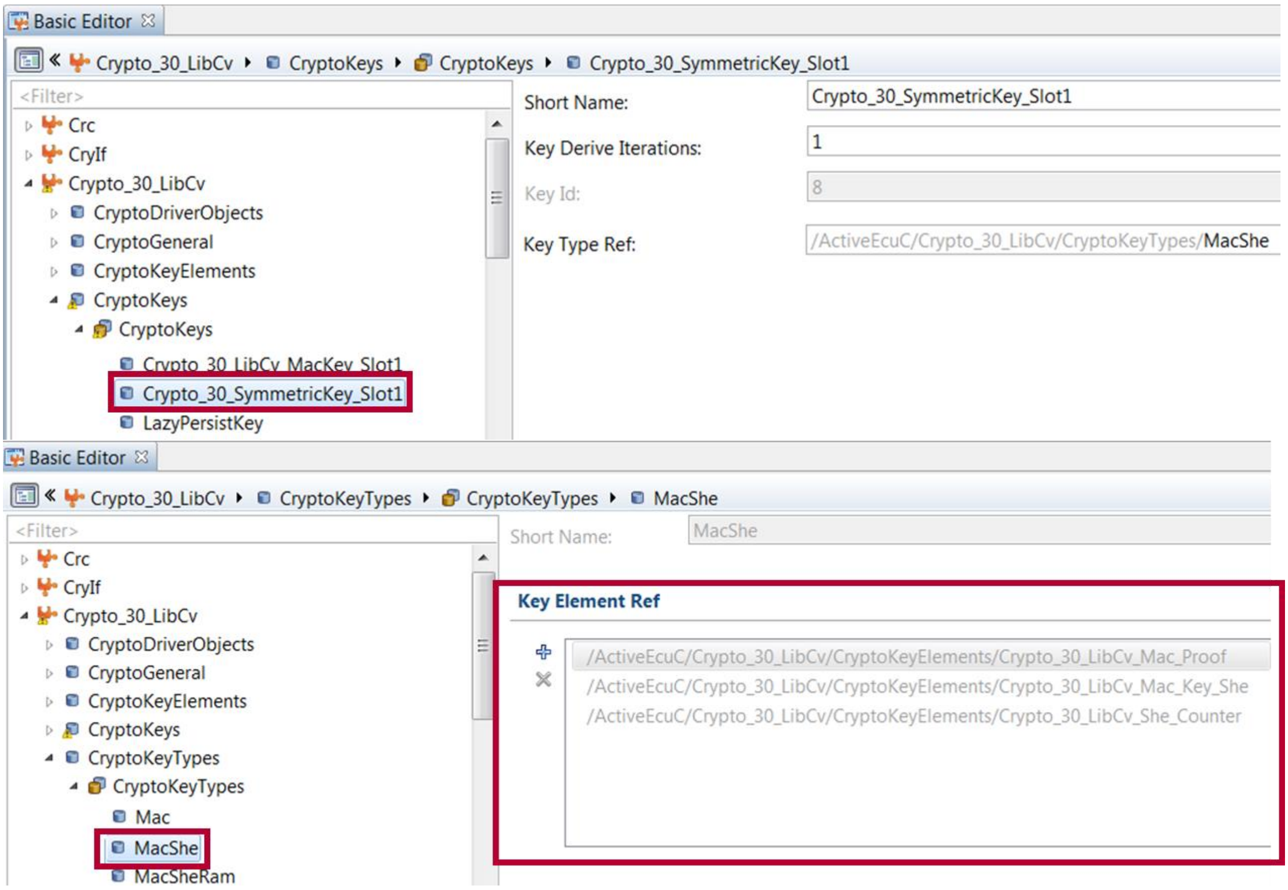

Figure 4-9 Normal SHE Key to Key Element Mapping

The screenshot above shows the Key Type Configuration which abstracts key configuration and respective key elements configuration.

It shows that the key mackey_Slot1 has three key elements.

4.5.1 SHE_Key (Which is the actual SHE key)

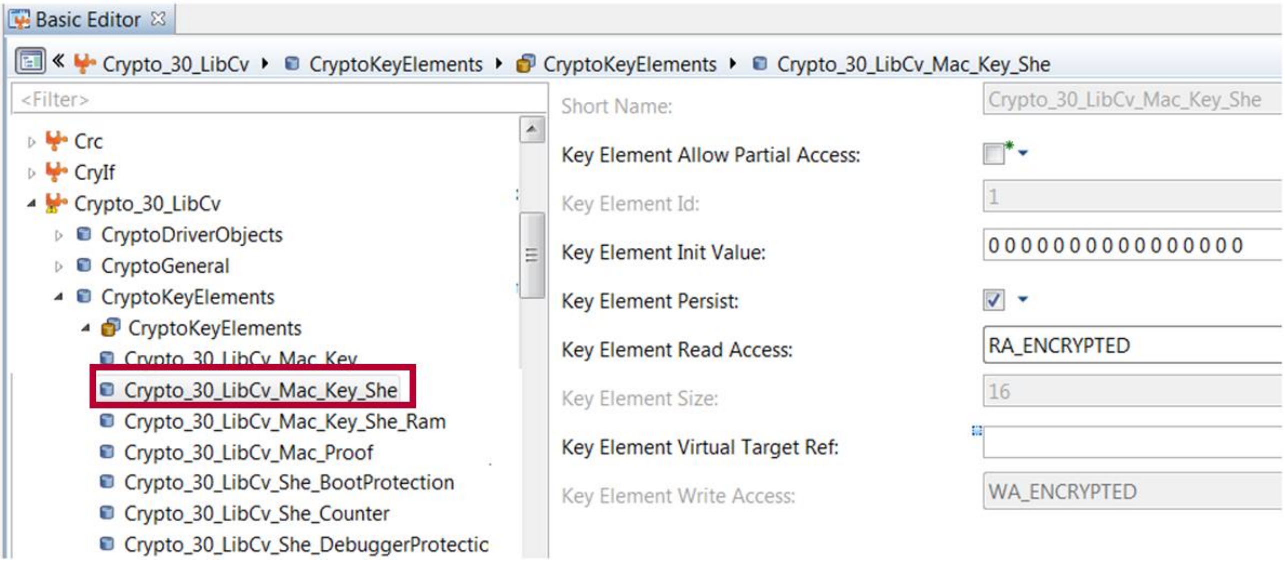

Figure 4-10 Actual SHE Key Element Configuration

> **Key Element Init Value**: These init values are being used by the software, in case a key element was not persisted or issue during reading of persisted values occur.

> **Key Element Persist**: **This option will be used by the Crypto SW Library, where the keys are writing in the NVM. If this option is enabled, then only this key will be written in the NVM during NVM_WriteAll function**.

> For the SHE Key Read Access & Write Access is always WA_ENCRYPTED.

4.5.2 SHE Key Proof

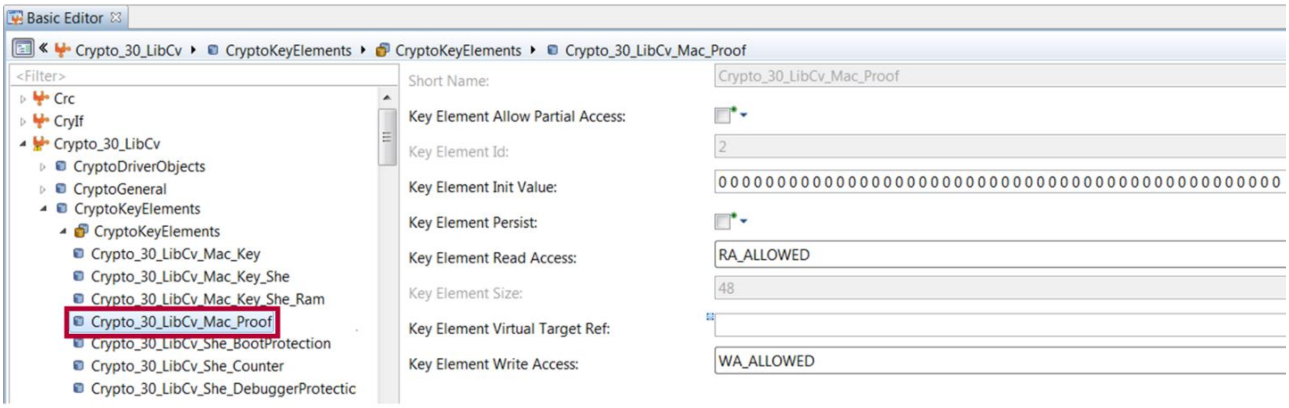

Figure 4-11 SHE Key Proof (Key Element) Configuration

Poof key element is being used to store the M4 & M5 values, which is of size 48 bytes. Refer section 3.6 & 3.7. Generally proof (M4 & M5) won’t be persisted in the NVM. As mentioned in the section 3 (Figure 3-1), these values are needed only for the verification, which will be sent to the tester.

4.5.3 SHE Counter

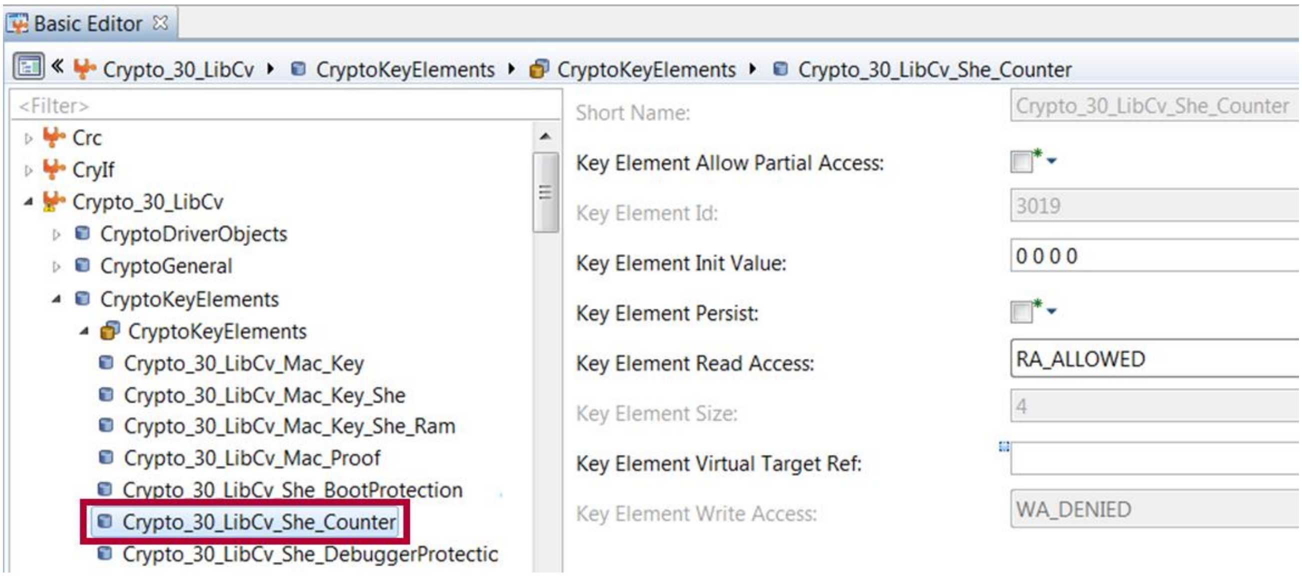

Figure 4-12 SHE Counter (Key Element) Configuration

Note
As per the section 4.2, if the SHE enable counter is enabled then this key element value needs to be stored. If user is using NVM option, then user has to enable **Key Element Persist**.

Reference
Refer section 3 Step 7 & 3.1 for the CID Functionality.

5 Flow Chart

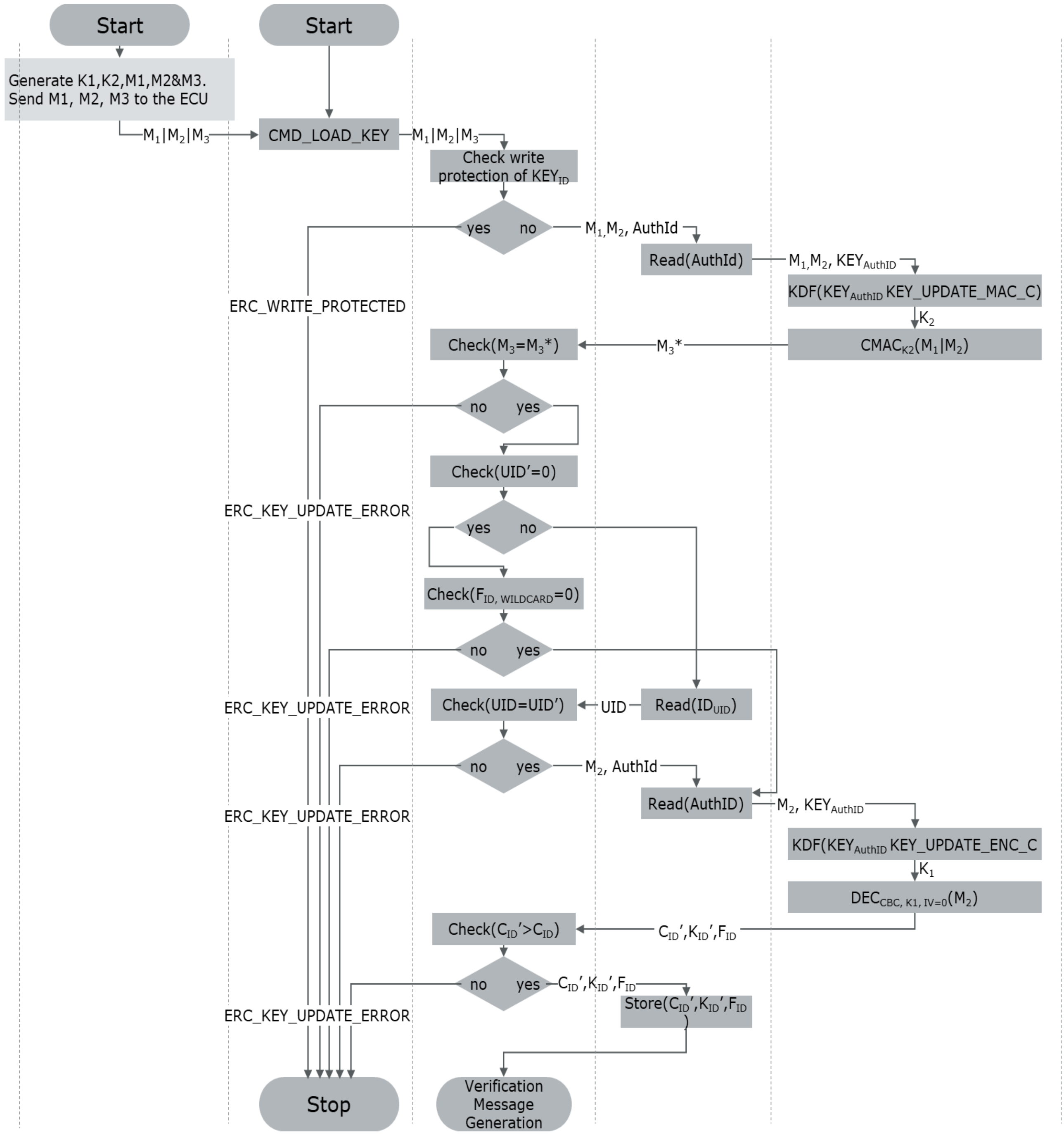

Figure 5-1 Simple Flow Chart to Persist the SHE Key

6 Code Flow

Note
To persist any SHE Key, the user has to call: `Csm_KeyElementSet (keyId, keyelementID, m1m2m3, 64)`. Once the key has been written, the user has to call `Csm_KeySetValid (keyId)`. Key can be used only after the above two functions are successful.

64 Bytes:→ Length of m1m2m3, m1m2m3 :→ Pointer to the received data of M1 M2 M3 values

```
/* Key Management Function Prototypes */
/**********************************************************************************************************
 *  Csm_KeyElementSet()
 **********************************************************************************************************/
/*! \brief         Sets the given key element bytes to the key identified by keyId.
 *  \details       -
 *  \param[in]     keyId                   Holds the identifier of the key for which a new material shall be set.
 *  \param[in]     keyElementId            Holds the identifier of the key element to be written.
 *  \param[in]     keyPtr                  Holds the pointer to the key element bytes to be processed.
 *  \param[in]     keyLength               Contains the number of key element bytes.
 *  \return        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 because write access was denied.
 *                 CRYPTO_E_KEY_NOT_AVAILABLE Request failed because the key is not available
 *                 CRYPTO_E_KEY_SIZE_MISMATCH  Request failed, key element size does not match size of provided data.
 *  \context       TASK
 *  \reentrant     TRUE, but not for the same keyId
 *  \synchronous   TRUE
 *  \pre           -
 *  \trace         SPEC-2820439, SPEC-2820440
 **********************************************************************************************************/
FUNC(Std_ReturnType, CSM_CODE) Csm_KeyElementSet(uint32 keyId,
                                                 uint32 keyElementId,
                                                 P2CONST(uint8, AUTOMATIC, CSM_APPL_VAR) keyPtr,
                                                 uint32 keyLength);

/**********************************************************************************************************
 *  Csm_KeySetValid()
 **********************************************************************************************************
/*! \brief         Sets the key state of the key identified by keyId to valid.
 *  \details       -
 *  \param[in]     keyId         Holds the identifier of the key for which a new material shall be validated.
 *  \return        E_OK          Request successful
 *                 E_NOT_OK      Request failed
 *                 CRYPTO_E_BUSY Request failed, Crypto Driver Object is busy
 *  \context       TASK
 *  \reentrant     TRUE, but not for the same keyId
 *  \synchronous   TRUE
 *  \pre           -
 *  \trace         SPEC-2820441, SPEC-2820442
 **********************************************************************************************************
FUNC(Std_ReturnType, CSM_CODE) Csm_KeySetValid(uint32 keyId);
```

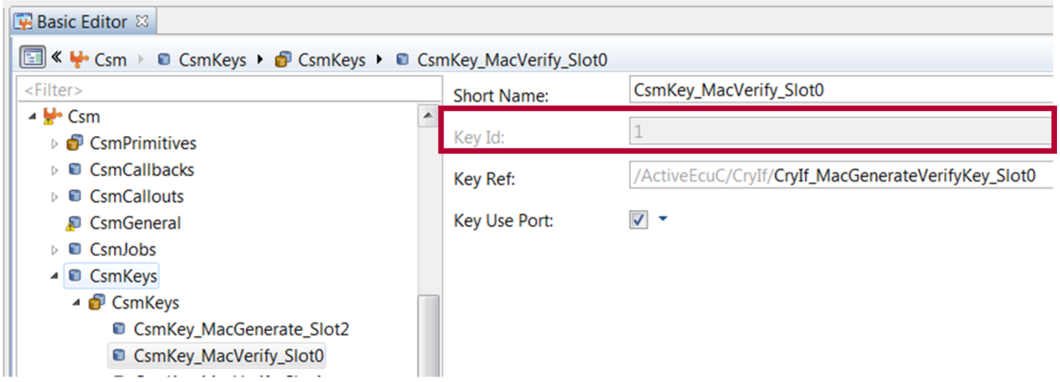

Figure 6-1 Prototype of Csm_KeyElementSet, Csm_KeySetValid

Key Id is known from the respective CSM Key configuration

Key Element ID is known from the respective key element configuration from the Crypto Driver configuration. In this case key is the actual SHE key which has to be persisted. Refer section 4.5.1.

6.1 Call Stack from CSM (Csm_KeyElementset)

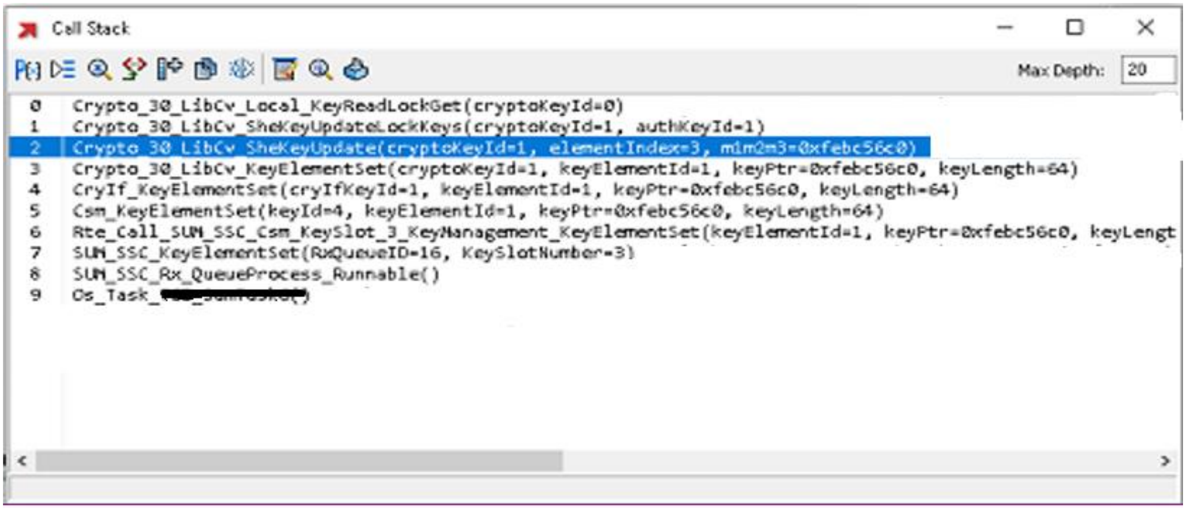

Figure 6-2 Call Stack of the Csm_KeyElementSet Function

`Csm_KeyElementSet` function calls the `Cryif_KeyElementSet` function. Based on the key mapping to the respective crypto drivers (SW Library or vHsm), the respective key element set function will be called.

This Crypto Driver function be selected from the below structure in the CryIf_Cfg.c file

```
CONST (CryIf_CryptoFunctionsType, CRYIF_CONST)
CryIf_CryptoFunctions[x] = {}
```

```
KeyElementSet
Crypto_30_vHsm_Hal_KeyElementSet    ,
Crypto_30_vHsm_Core_KeyElementSet   ,
Crypto_30_LibCv_KeyElementSet       ,
Crypto_30_vHsm_Custom_KeyElementSet,
```

Figure 6-3 Structure of CryIf_CryptoFunctions

```
/**********************************************************************************************************************
 *  Crypto_30_LibCv_KeyElementSet()
 *********************************************************************************************************************/
/*! \brief         Sets a key element
 *  \details       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.
 *                 The access right need to be WA_ALLOWED for normal key, WA_ENCRYPTED for She key and WA_ALLOWED for She ram key
 *  \param[in]     cryptoKeyId            Holds the identifier of the key whose key element shall be set.
 *  \param[in]     keyElementId           Holds the identifier of the key element which shall be set.
 *  \param[in]     keyPtr                 Holds the pointer to the key data which shall be set as key element.
 *  \param[in]     keyLength              Contains the length of the key element in bytes.
 *  \return        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.
 *  \pre           -
 *  \context       TASK
 *  \reentrant     TRUE, for different crypto keys
 *  \synchronous   TRUE
 *********************************************************************************************************************/
FUNC(Std_ReturnType, CRYPTO_30_LIBCV_CODE) Crypto_30_LibCv_KeyElementSet(
  uint32 cryptoKeyId,
  uint32 keyElementId,
  P2CONST(uint8, AUTOMATIC, CRYPTO_30_LIBCV_APPL_VAR) keyPtr,
  uint32 keyLength);
```

Figure 6-4 Prototype of Crypto_30_Libcv_KeyElementSet

The function above checks for the write protection and returns `CRYPTO_E_KEY_NOT_AVAILABLE` in case if the write protection has been configured different from `WA_ENCRYPTED` for the SHE Key.

6.2 Crypto_30_LibCv_SheKeyUpdate

In both cases of vHsm & SW Crypto, function `Crypto_30_LibCv_SheKeyUpdate` will be called, after write protection verifies.

```
/*********************************************************************************************************************
 * Crypto_30_LibCv_SheKeyUpdate()
 *********************************************************************************************************************/
/*! \brief         Updates key element based on SHE key update mechanism.
 *  \details       Interprets the given key buffer as M1M2M3 of the SHE key update mechanism and extracts relevant
 *                 information for setting the key element. This is used during the SHE key updated protocol.
 *  \param[in]     cryptoKeyId            Holds the identifier of the key whose key element shall be set.
 *  \param[in]     elementIndex           Holds the identifier of the key element which shall be set
 *  \param[in]     indexOfSheKey          Hold the index of the update key in SheKeys.
 *  \param[in]     m1m2m3                 Holds the pointer to the key buffer which shall be used to update the key element which holds M1M2M3.
 *  \return        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.
 *  \pre           cryptoKeyId, elementIndex as well as indexOfSheKey must identify a valid key - key element pair
 *                 keyPtr has to be a valid ptr with the length of CRYPTO_30_LIBCV_SIZEOF_SHE_M1_M3
 *  \context       TASK
 *  \reentrant     TRUE, for different crypto keys.
 *  \synchronous   TRUE
 *********************************************************************************************************************/
CRYPTO_30_LIBCV_LOCAL_INLINE FUNC(Std_ReturnType, CRYPTO_30_LIBCV_CODE) Crypto_30_LibCv_SheKeyUpdate(
  uint32 cryptoKeyId,
  Crypto_30_LibCv_SizeOfKeyElementsType elementIndex,
  Crypto_30_LibCv_SizeOfSheKeysType indexOfSheKey,
  P2CONST(uint8, AUTOMATIC, CRYPTO_30_LIBCV_APPL_VAR) m1m2m3);
```

```
CRYPTO_30_LIBCV_LOCAL_INLINE FUNC(Std_ReturnType, CRYPTO_30_LIBCV_CODE) Crypto_30_LibCv_SheKeyUpdate(
  uint32 cryptoKeyId,
  Crypto_30_LibCv_SizeOfKeyElementsType elementIndex,
  P2CONST(uint8, AUTOMATIC, CRYPTO_30_LIBCV_APPL_VAR) m1m2m3)
{
  Std_ReturnType retVal = E_NOT_OK;
  Std_ReturnType proofAvailable = E_NOT_OK;
  Crypto_30_LibCv_KeyElementsIterType outputElement;

  Crypto_30_LibCv_SizeOfSheKeysType indexOfSheKey, indexOfAuthSheKey;

  if (Crypto_30_LibCv_SheKeyGetSheIndex(elementIndex, &indexOfSheKey) == E_OK) /* SBSW_CRYPTO_30_LIBCV_STACK_VARIABLE_AS_PTR */
  {
    /* check write protection */
# if (CRYPTO_30_LIBCV_SHE_ENABLE_FID == STD_ON)
    if (!Crypto_30_LibCv_IsKeyElementStateByMask(elementIndex, CRYPTO_30_LIBCV_KEYELEMENTSTATE_WRITTEN_ONCE_MASK))
# endif
    {
      /* # Determine proof output slot */
      proofAvailable = Crypto_30_LibCv_SheKeyUpdateFindProof(cryptoKeyId, &outputElement); /* SBSW_CRYPTO_30_LIBCV_STACK_VARIABLE_AS_PTR */

      if ((proofAvailable == E_OK) || (proofAvailable == CRYPTO_E_KEY_NOT_AVAILABLE))
      {
        /*# Check key ID */
        if (Crypto_30_LibCv_SheKeyUpdateCheckM1Ids(m1m2m3, indexOfSheKey, &indexOfAuthSheKey) == E_OK) /* SBSW_CRYPTO_30_LIBCV_STACK_VARIABLE_AS_PTR */
        {
          if (Crypto_30_LibCv_SheKeyUpdateLockKeys(cryptoKeyId, Crypto_30_LibCv_GetKeyIdxOfSheKeys(indexOfAuthSheKey)) == E_OK)
          {
          /* # Verify M3, Extract Key and Generate M4 & M5, if proof slot is available  */
            retVal = Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract(elementIndex, m1m2m3, indexOfSheKey, indexOfAuthSheKey, (Crypto_30_LibCv_SizeOfKeyEle

            Crypto_30_LibCv_SheKeyUpdateUnlockKeys(cryptoKeyId, Crypto_30_LibCv_GetKeyIdxOfSheKeys(indexOfAuthSheKey));
          }
          else
          {
            retVal = CRYPTO_E_BUSY;
          }
```

Get the SHE Index from the structure Crypto_30_LibCv_SheKeys

This is ON when FID option has been enabled. Refer Section 4.2

Checking for WRITE_PROTECTION flag

Checking for availability of Proof Key Element Slot. Refer to Section 4.5.2

Figure 6-5 Crypto_30_LibCv_SheKeyUpdate Function

6.3 Crypto_30_LibCv_SheKeyUpdateCheckM1Ids

> This function extracts the both, Authentication ID and SHE ID from the M1 value (Refer section 3.3).

> SHE ID will be verified with configured SHE ID. Configured SHE ID is available in the structure **Crypto_30_LibCv_SheKeys**.

> Send E_NOT_OK if the received SHE ID not identified in the configuration.

> Send E_NOT_OK if the SHE ID is SECRET Key ID.

> Send E_OK if the SHE ID is same as Authentication ID.

> If SHE ID is not same as Authentication ID, check the Authentication ID as MASTER KEY ID, if yes extract the Authentication ID index details.

6.4 Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract

```
/**********************************************************************************************************
 *  Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract()
 **********************************************************************************************************/
/*! \brief         Verify and extract key
 *  \details       Verify M3, extract key and generate M4 and M5.
 *  \param[in]     elementIndex            Holds the identifier of the key element which shall be set
 *  \param[in]     m1m2m3                  Holds the pointer to the key buffer which shall be used to update the key element with M1M2M3.
 *  \param[in]     indexOfSheKey           Index of the update key element in SheKeys
 *  \param[in]     indexOfAuthSheKey       Index of the authentication key element in SheKeys
 *  \param[in]     outputElement           Index of the destination key element
 *  \param[in]     proofAvailable          Holds the value if the proof element is available.
 *  \return        E_OK                    Request successful.
 *                 E_NOT_OK                Request failed.
 *  \pre           cryptoKeyId and outputElement as well as elementIndex must identify a valid key - key element pair
 *                 keyPtr has to be a valid ptr with the length of CRYPTO_30_LIBCV_SIZEOF_SHE_M1_M3
 *                 indexOfSheKey and indexOfAuthSheKey need to be valid index for SheKeys.
 *  \context       TASK
 *  \reentrant     TRUE, for different crypto keys.
 *  \synchronous   TRUE
 **********************************************************************************************************/
CRYPTO_30_LIBCV_LOCAL_INLINE FUNC(Std_ReturnType, CRYPTO_30_LIBCV_CODE) Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract(
  Crypto_30_LibCv_SizeOfKeyElementsType elementIndex,
  P2CONST(uint8, AUTOMATIC, CRYPTO_30_LIBCV_APPL_VAR) m1m2m3,
  Crypto_30_LibCv_SizeOfSheKeysType indexOfSheKey,
  Crypto_30_LibCv_SizeOfSheKeysType indexOfAuthSheKey,
  Crypto_30_LibCv_SizeOfKeyElementsType outputElement,
  Std_ReturnType proofAvailable);
```

```
CRYPTO_30_LIBCV_LOCAL_INLINE FUNC(Std_ReturnType, CRYPTO_30_LIBCV_CODE) Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract(
  Crypto_30_LibCv_SizeOfKeyElementsType elementIndex,
  P2CONST(uint8, AUTOMATIC, CRYPTO_30_LIBCV_APPL_VAR) m1m2m3,
  Crypto_30_LibCv_SizeOfSheKeysType indexOfSheKey,
  Crypto_30_LibCv_SizeOfSheKeysType indexOfAuthSheKey,
  Crypto_30_LibCv_SizeOfKeyElementsType outputElement,
  Std_ReturnType proofAvailable)
{
  /* # Init KDF Parameters */
  /* KDF Buffer */
  uint8 KDFbuffer[CRYPTO_30_LIBCV_SIZEOF_KDF_BUFFER];

  /* # Init Workspace */
  /* Working Buffers */
  uint8 encBuffer[CRYPTO_30_LIBCV_SIZEOF_ENC_BUFFER];

  Std_ReturnType retVal = E_NOT_OK;
  Crypto_30_LibCv_SheKeyTypeType sheKeyType = Crypto_30_LibCv_SheKeyGetKeyType(Crypto_30_LibCv_GetSheIdOfSheKeys(indexOfSheKey));
                                  This function will be explained below
  /* # Verify M3 */
  if (Crypto_30_LibCv_SheKeyUpdateVerifyM3(m1m2m3, indexOfAuthSheKey, indexOfSheKey, KDFbuffer, CRYPTO_30_LIBCV_SIZEOF_KDF_BUFFER, encBuffer) == E_OK) /* SBSW_CRYPT
  {
    /* # check UID */               Check the received UID against the wild card value or stored ECU ID, refer section 3.1
    if (Crypto_30_LibCv_SheKeyUpdateCheckUid(m1m2m3, elementIndex, sheKeyType) == E_OK) /* SBSW_CRYPTO_30_LIBCV_FORWARDING_OF_KEYPTR */
    {
      /* # Extract Key */
      retVal = Crypto_30_LibCv_SheKeyUpdateExtractKey(m1m2m3, elementIndex, KDFbuffer, CRYPTO_30_LIBCV_SIZEOF_KDF_BUFFER, encBuffer, indexOfSheKey); /* SBSW_CRYPTO_

      if (retVal == E_OK)                 This function will be explained below
      {
        /* store FID and Counter */
# if(CRYPTO_30_LIBCV_SHE_ENABLE_COUNTER == STD_ON)
        /* RAM key has no counter */
        if (Crypto_30_LibCv_IsKeyElementsCounterUsedOfSheKeys(indexOfSheKey))
        {
          Crypto_30_LibCv_SheKeyUpdateCopyCounter(Crypto_30_LibCv_GetKeyElementsCounterIdxOfSheKeys(indexOfSheKey), encBuffer); /* SBSW_CRYPTO_30_LIBCV_STACK_VARIAE
        }
# endif                                                                                                          Store the received counter
# if (CRYPTO_30_LIBCV_SHE_ENABLE_FID == STD_ON)                                                                  and FID values in the
        Crypto_30_LibCv_SheKeyUpdateCopyFid(elementIndex, encBuffer, sheKeyType); /* SBSW_CRYPTO_30_LIBCV_STACK_VARIABLE_AS_PTR */   respective key elements
# else
        Crypto_30_LibCv_ClearKeyElementExtensionByMask(elementIndex, CRYPTO_30_LIBCV_KEYELEMENTSEXTENSION_SHE_CLEAR_PLAIN_KEY_MASK); /* SBSW_CRYPTO_30_LIBCV_CSL02_F
# endif

        /* # Generate M4 & M5, if proof slot is available */
        if (proofAvailable == E_OK)
        {
          retVal = Crypto_30_LibCv_SheKeyUpdateProofM4M5(m1m2m3, (Crypto_30_LibCv_SizeOfKeyElementsType)outputElement, KDFbuffer, encBuffer, indexOfSheKey); /* SBSW
        }
      }                       This function generates M4, M5 and store in the Proof Key element
    }
  }

  return retVal;
} /* PRQA S 6060 */ /* MD_MSR_STPAR */
```

Figure 6-6 Crypto_30_LibCv_SheKeyUpdateVerifyAndExtract Function

6.5 Crypto_30_LibCv_SheKeyUpdateVerifyM3

Note
Only in vHsm checking for the Authentication key validity, in Crypto SW library this check is not there.

> Update the Key_Update_Mac_C from the structure **Crypto_30_LibCv_SheConstants**
> Calculate K2 by using AES-MP method `Crypto_30_LibCv_SheKeyUpdateMiyaguchiPreneel ()`
> Calculate the CMAC
> Compare the generated CMAC with M3 value
> Return E_OK if matches, E_NOT_OK if is not matched.

6.6 Crypto_30_LibCv_SheKeyUpdateExtractKey

> After “M3 and UID” verification successful, need to extract the SHE Key, UID, CID from M2.
> Update **Key_Update_Enc_C** from the structure `Crypto_30_LibCv_SheConstants`
> Calculate K1 by using AES-MP method `Crypto_30_LibCv_SheKeyUpdateMiyaguchiPreneel ()`
> Decrypt the M2 Data, which is UID, FID and SHE Key Value.
> If Enable Counter is ON (CRYPTO_30_LIBCV_SHE_ENABLE_COUNTER == STD_ON, refer section 4.2 for the respective configuration parameter), received counter value from M2 will be verified with the stored counter value, which is extracted from the respective Key Element ID.

Note
If the user has enabled the **Enable SHE Counter**, the user has to configure one key element for counter. Refer 4.2, 4.5, 4.5.3.

> New counter value from M2 should be greater than stored counter value
> Store the new SHE key in the respective buffer in the respective index of the array `Crypto_30_LibCv_KeyStorage`

6.7 Csm_KeyElementSetValid Handling

- `Csm_KeySetValid` calls `CryIf_KeySetValid`.
- `CryIf_KeySetValid` calls `Crypto_30_LibCv_KeyValidSet` in case of Crypto SW through the `CryIf_CryptoFunctions`
- Below are possible valid functions in the `CryIf_CryptoFunctions` (includes vHsm)

```
KeyValidSet
Crypto_30_vHsm_Hal_KeyValidSet    ,
Crypto_30_vHsm_Core_KeyValidSet   ,
Crypto_30_LibCv_KeyValidSet       ,
Crypto_30_vHsm_Custom_KeyValidSet,
```

Figure 6-7 Possible key Validity Functions in the Crypto Driver

- `Crypto_30_LibCv_KeyValidSet` function calls `Crypto_30_LibCv_SetKeyState`
- `Crypto_30_LibCv_SetKeyState` sets the validity bit in all respective mapped key elements

Note
To store the validity bit, one byte has been allocated for each key element. This validity status (this byte) will be stored in the array Crypto_30_LibCv_KeyStorage, in the respective Index.

Index details can be known through the Crypto_30_LibCv_KeyElements structure.

Refer appendix Figure 7-4 & Figure 7-5.

7 Appendix

7.1 Appendix A

7.1.1 How to Get Stored Key Details

> All key values will be stored in the variable VAR (`Crypto_30_LibCv_KeyStorageType`, `CRYPTO_30_LIBCV_VAR_NOINIT`) `Crypto_30_LibCv_KeyStorage []`

> This Array is available in the Crypto_Libcv_30_cfg.c file

> The index details of a particular key (representation of each byte in the `Crypto_30_LibCv_KeyStorage`) will be known through the structure `CONST (Crypto_30_LibCv_KeyElementsType, CRYPTO_30_LIBCV_CONST) Crypto_30_LibCv_KeyElements []`

> In case of Crypto SW Library, below is the `Crypto_30_LibCv_KeyElements` structure format

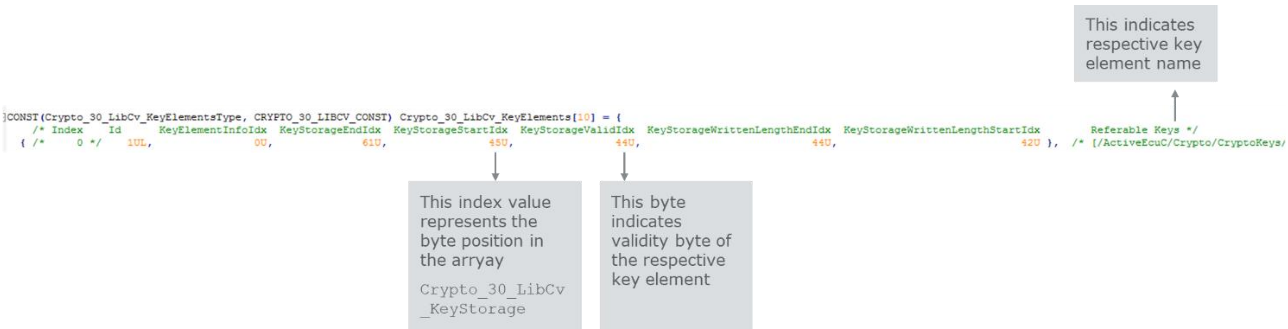

Figure 7-1 Crypto_30_LibCv_KeyElements Structure Format (Crypto SW)

In case of vHsm, below is the `Crypto_30_LibCv_KeyElements` structure format

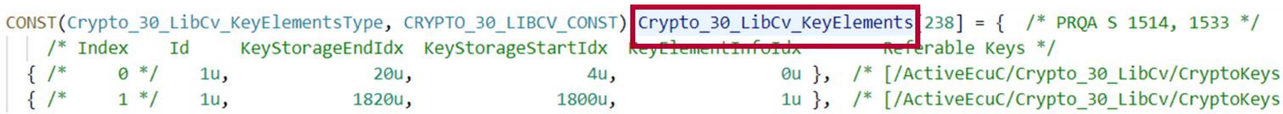

Figure 7-2 Crypto_30_LibCv_KeyElements Structure Format (vHsm)

For the other elements (Valid Index, Length Start Index and End Index), it follows calculation below.

Index information is extracted from the start index. Byte order is same as Crypto SW.

```
#define Crypto_30_LibCv_GetKeyStorageExtensionIdxOfKeyElements(Index) ((Crypto_30
_LibCv_KeyStorageExtensionIdxOfKeyElementsType)((Crypto_30_LibCv_GetKeyStorageStartIdxOfKeyElements(Index) - 1u))) /**< the
index of the 1:1 relation pointing to Crypto_30_LibCv_KeyStorage */

#define Crypto_30_LibCv_GetKeyStorageValidIdxOfKeyElements(Index) ((Crypto_30_LibCv_KeyStorageValidIdxOfKeyElementsType)
((Crypto_30_LibCv_GetKeyStorageStartIdxOfKeyElements(Index) - 2u))) /**< the index of the 1:1 relation pointing to Crypto_30
_LibCv_KeyStorage */

#define Crypto_30_LibCv_GetKeyStorageWrittenLengthEndIdxOfKeyElements(Index) ((Crypto_30
_LibCv_KeyStorageWrittenLengthEndIdxOfKeyElementsType)((Crypto_30_LibCv_GetKeyStorageStartIdxOfKeyElements(Index) - 2u))) /
**< the end index of the 1:n relation pointing to Crypto_30_LibCv_KeyStorage */


#define Crypto_30_LibCv_GetKeyStorageWrittenLengthStartIdxOfKeyElements(Index) ((Crypto_30
_LibCv_KeyStorageWrittenLengthStartIdxOfKeyElementsType)((Crypto_30_LibCv_GetKeyStorageStartIdxOfKeyElements(Index) - 4u))) /
**< the start index of the 1:n relation pointing to Crypto_30_LibCv_KeyStorage */
```

Figure 7-3 Calculation to Extract Valid Index, Start Index , End Index (vHsm)

7.1.2 Crypto NVM Block Configuration

> In case of Crypto SW Library and if the user has opted to store the keys in the NVM, the user has to configure as in the screenshot below.

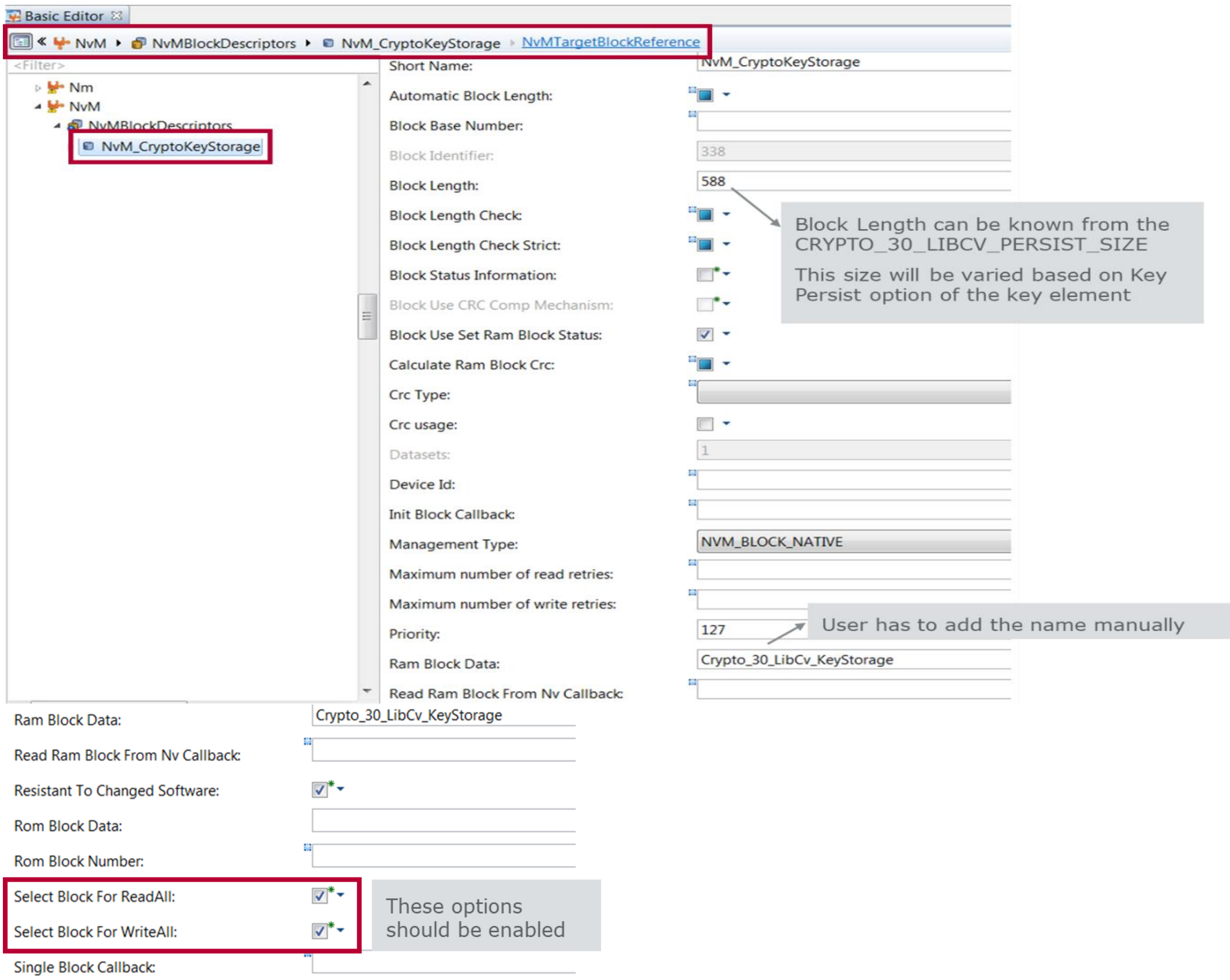

Figure 7-4 Crypto NVM Block Configuration

During NVM Write Crypto_30_Libcv_KeyStorage will be written into the NVM and during NVM read all data from the NVM will be copied to the `Crypto_30_Libcv_KeyStorage`.

Hints for Crypto Init

To ensure the correct restoring of the persisted key, keep the initialization in the correct order:

> `Crypto_30_LibCv_InitMemory (INIT_MEMORY)`

> `Crypto_30_LibCv_Init (INIT_TWO_DRV)`

> `NvM_ReadAll`

The CRYPTO module does not handle its configured NVM block explicitly. Therefore, the NVM block needs to be read manually or during the `NvM_ReadAll` operation. For persisting keys, the block needs either be written manually or needs to be included in the `NvM_WriteAll` operation.

The CRYPTO provides an optional feature to call `NvM_SetRamBlockStatus`, whenever a key which needs to be persisted is validated. If a key is validated can be checked in Table 3-1. If the feature is disabled, the CRYPTO module does not mark a block as modified (`NvM_SetRamBlockStatus`), it is up to the NMV to detect the need of writing the block.

7.2 Appendix B

Note

Below stack flow for few uses cases, which would give idea about the function flow.

The flow will vary based on customer requirements and configuration.

Especially for the random number generation.

The flow would be same until dispatch service and further function calls may vary based on the configuration.

7.2.1 Call Stack of MAC Generate & Verify

```
0   esl_initCMACAES128(workSpace=0xfeffb304, keyLength=64,
1   Crypto_30_LibCv_InitializeCmacAes(objectId=0, job=0xfeff9298, workspace=
2   Crypto_30_LibCv_DispatchCmacAesGenerate(objectId=0, job=0xfeff9298, mode
3   Crypto_30_LibCv_DispatchMacGenerate(objectId=0, job=0xfeff9298, mode=1)
4   Crypto_30_LibCv_DispatchService(objectId=0, job=0xfeff9298, mode=1)
5   Crypto_30_LibCv_Process(objectId=0, job=0xfeff9298)
6   Crypto_30_LibCv_ProcessJob(objectId=0, job=0xfeff9298)
7   CryIf_ProcessJob(channelId=0, job=0xfeff9298)
8   Csm_HandleJobProcessing(channelId=0, job=0xfeff9298)
9   Csm_ProcessJob(queueIdx=0, job=0xfeff9298)
10  Csm_MacGenerate(jobId=0, mode=7, dataPtr=0xfefe9c73, dataLength=14, macP
11  Rte_Call_SUM_SSC_Csm_KeySlot_2_MacGenerate_MacGenerate(dataBuffer=0xfefe
12  SUM_SSC_Call_MacGenerate(TxQueueID=26)
13  SUM_SSC_Tx_QueueProcess_Runnable()
```

Figure 7-5 Call stack of MAC Generate

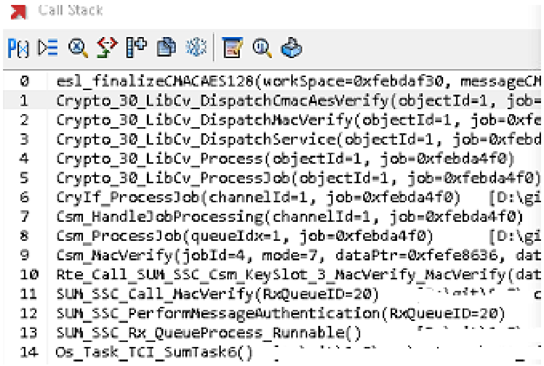

Figure 7-6 Call Stack of MAC Verify

7.2.2 Call Stack of Encrypt, Decrypt & Random Generate

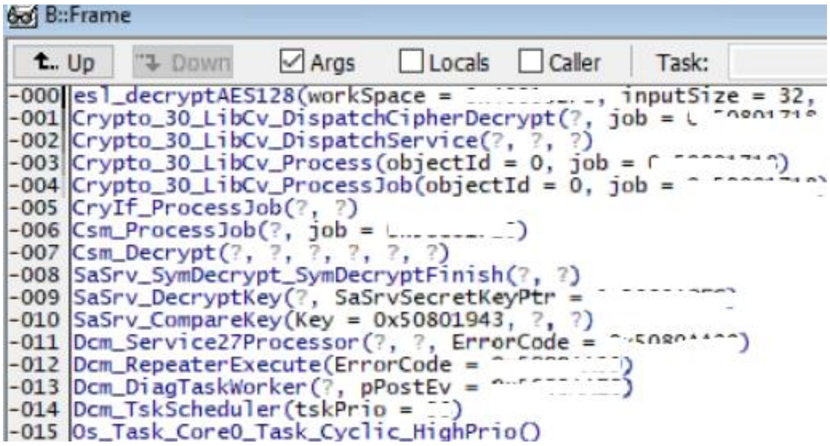

Figure 7-7 Call Stack of 27 Service Decrypt Key (AES Decrypt)

Figure 7-8 Call Stack of 27 Service Encrypt Seed (AES Encrypt)

Below is the call stack.

```
27 01
SaSrv_GetSeed()
SaSrv_SecretSeed()
SaSrv_RandomGenerate_RandomGenerate()
Csm_ProcessJob()
CryIf_ProcessJob()
CryIf_GetProcessJobOfCryptoFunctions()
Crypto_30_LibCv_DispatchService()
Crypto_30_LibCv_DispatchRandom()
Crypto_30_LibCv_DispatchRandomNistDrbgAes()
Crypto_30_LibCv_DispatchService()
Crypto_30_LibCv_DispatchRandomNistDrbgAesFinish()
Crypto_30_LibCv_AesCtrDrbgGenerateRn()
```

```
Csm_RandomSeed
CryIf_RandomSeed
Crypto_30_LibCv_RandomSeed
Crypto_30_LibCv_Local_RandomSeed
Crypto_30_LibCv_Local_RandomSeed_NistDrbgAes
Crypto_30_LibCv_AesCtrDrbgSeed
Crypto_30_LibCv_AesCtrDrbgReseed
Crypto_30_LibCv_AesCtrDrbgInstantiation
Crypto_30_LibCv_AesCtrDrbgDF
```

Figure 7-9 Call Stack of 27 Service Random Number Generation

8 Glossary and Abbreviations

8.1 Abbreviations

Abbreviation	Description
AUTOSAR	Automotive Open System Architecture
AES	Advanced Encryption Standard
AES-MP	Advanced Encryption Standard - Miyaguchi-Preneel
Auth ID	Authentication Identifier
BP	Boot Protection
CID	Counter Identification
CMAC	Cipher Based Message Authentication Code
Cry IF	Crypto Interface
CSM	Crypto Service Manager
CU	CMAC Usage
DP	Debug Protection
ECU	Electronic Control Unit
ENC	Encryption
FID	Flag Identification (Protection Flags)
K1	Key “1” (numeric number may vary)
KU	Key Usage (0=AES, 1=CMAC)
MAC	Message Authentication Code
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
NVM	Non-Volatile Memory
RA_	Read Access
SHE	Secure Hardware Extension
SWS	Software Specification
UID	Unique Identification Identifier
WA_	Write Access
WC	Wild Card
WP	Write Protection

9 Contact

Visit our website for more information on

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

www.vector.com