NXP Semiconductors
Application Notes

Document Number: AN 13061
Rev. 0, 12/2020

Using the CSE (Cryptographic Services Engine) Module in MCAL4.3

by: NXP Semiconductors

1 Introduction

Today, people are concerned about sharing/transmitting information in a secure and trusted manner between parties in the automotive area. The security functions implemented with Cryptographic Services Engine(CSE) module are described in the Secure Hardware Extension(SHE) functional specification.

This application note provides an introduction to the CSE module on the MPC5777C and explains how to configure and use this module in MCAL4.3. After reading this application note, you should be able to:

- Understand the CSE configuration process in MCAL4.3
- Understand how CSE module working on the MPC5777C

The application note also provides examples for specific configurations. The examples are provided in the software package accompanying this document and is also explained in detail. To aid in understanding of this document and software package, you need to download the MPC5777C reference manual from the NXP website. The following table shows the abbreviations used throughout the document.

Contents

Table 1. Abbreviations and acronyms

Abbreviation	Definition
CSE	Cryptographic Services Engine
CSM	Crypto Service Manager
CRYIF	Crypto Interface
Crypto	Crypto Driver

2 CSE module on MPC5777C

The Cryptographic Services Engine (CSE) is a peripheral module that implements the security functions described in the Secure Hardware Extension (SHE) Functional Specification Version 1.1.

The CSE design includes a host interface with a set of memory mapped registers and a system bus interface. The host interface are used by the CPU to issue commands. The system bus interface allows the CSE to access system memory directly. Two dedicated blocks of system flash memory are used by the CSE for secure key storage.

2.1 Chip-specific CSE information

This chip has one instance of the CSE module. The module:

- Executes the chip's secure boot process. See the System Boot details in the MPC5777C Reference Manual.
- Exclusive access to the flash memory blocks mapped to the C55FMC_LOCK1 register, PASS_LOCK1_PGn registers, and TDRn_LOCK1 DCF client. See C55FMC_LOCK1 register bit mapping. The system MPU is automatically configured to prevent other bus masters from interfering with CSE's access to the flash memory.

2.2 Features

The CSE has the following features:

- Secure storage for cryptographic keys
- AES-128 encryption and decryption
- AES-128 CMAC authentication
- True random number generation
- Secure boot mode
- System bus master interface

2.3 Modes of operation

The CSE supports operation in Normal and Debug modes of operation. The use of the cryptographic keys stored by the CSE is controlled based on the activation of the CPU debug port and the successful completion of the secure boot process.

The CSE has a low-power mode that disables the clock to all logic except the host interface. Register accesses are supported in this mode, but commands are not processed.

2.4 Block diagram

The CSE design includes a command processor, host interface, system bus interface, local memory, AES logic, and True Random Number Generator (TRNG) as shown below.

A host interface (via the peripheral bridge) with a set of memory mapped registers that are used by the CPU to issue commands. Furthermore, a system bus interface (via the crossbar interface) allows the CSE to directly access system memory. Here the crypto module behaves like any other master on the Crossbar switch (XBAR). Through the host interface, you can configure and control the CSE module, like putting the module into low power mode, enabling interrupts for finished command processing, or suspending command processing. A status and error register gives further system information. For a complete list of CSE commands see MPC5777C reference manual.

Two dedicated blocks of system flash memory are used by the CSE for secure key and firmware storage. These blocks are not accessible by other masters from the system. Therefore, they are called secure flash. The command processing is done by a 32-bit CSE core with attached ROM and RAM running at system frequency. After system boot, the core comes out of reset and executes boot code from the module ROM. This code will load the firmware from the secure flash into the module RAM and start executing from there. This reduces the flash accesses by the crypto core on the Crossbar. The AES block is a slave to the crypto internal bus. It processes the encryption (plaintext → ciphertext) and decryption (ciphertext → plaintext) and offers AES CMAC authentication. This application note deals only with the authentication capabilities of the CSE.

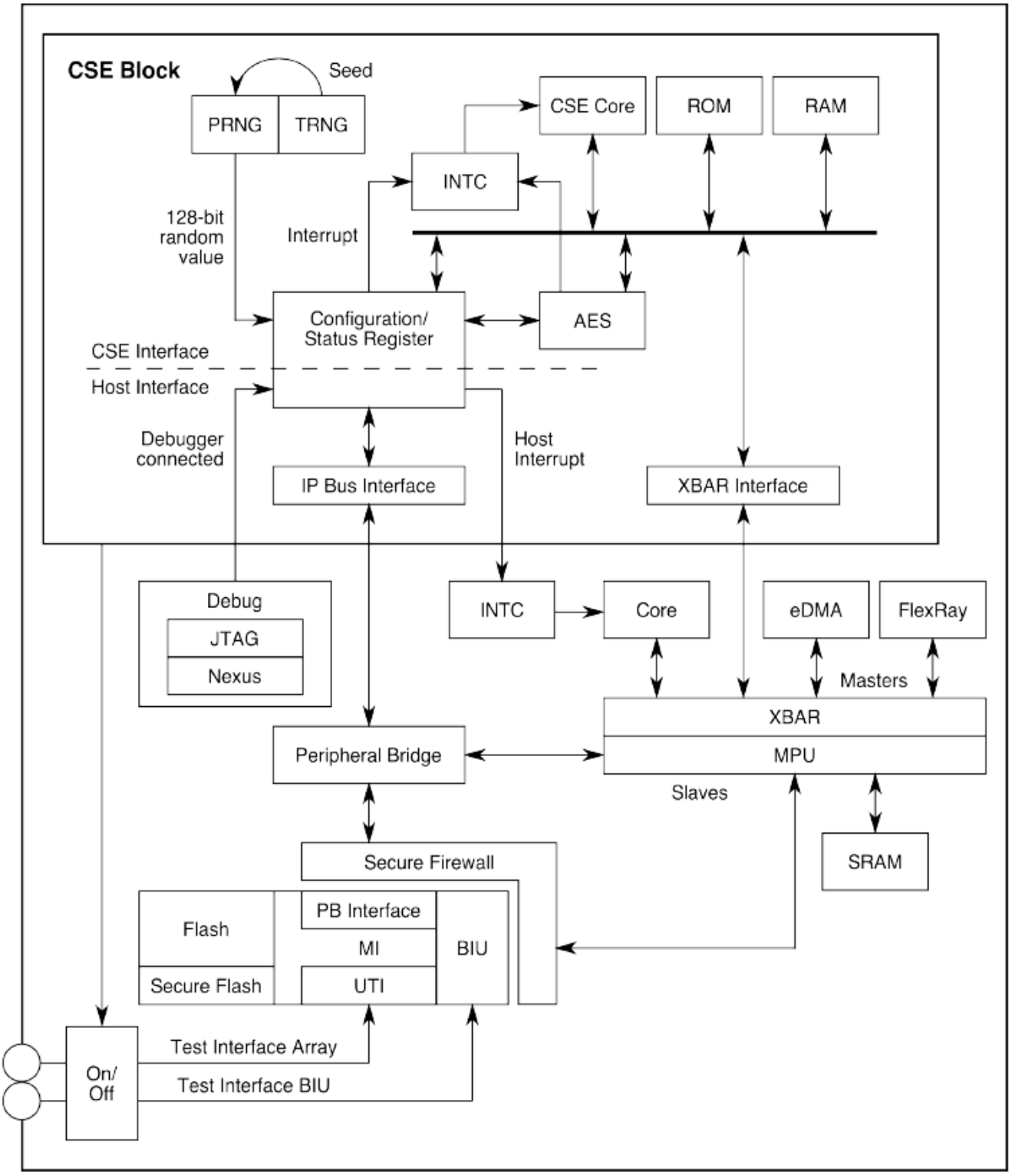

Figure 1. CSE block diagram on MPC5777C

3 AES-128 encryption and decryption overview

Block ciphers like the AES algorithm, working with a defined granularity, are often 64 bits or 128 bits. The simplest way to encode data is to split the message in the cipher specific granularity. In this case, the cipher output depends only on the key and input value. The drawback of this cipher mode, which is called Electronic Code Book (ECB), is that the same input values will be decoded into the same output values. This gives attackers the opportunity to use statistical analysis (for example, in a normal text some letter combinations occur much more often than others).

To overcome this issue other cipher modes were developed like the Cipher-block chaining (CBC), Cipher feedback (CFB), Output feedback (OFB) and Counter (CTR) mode.

The CSE module supports only the ECB and the CBC mode which are described in the following sections.

3.1 Electronic Codebook (ECB)

Each block has no relationship with another block of the same message or information. The following figure shows the block diagram of the ECB mode.

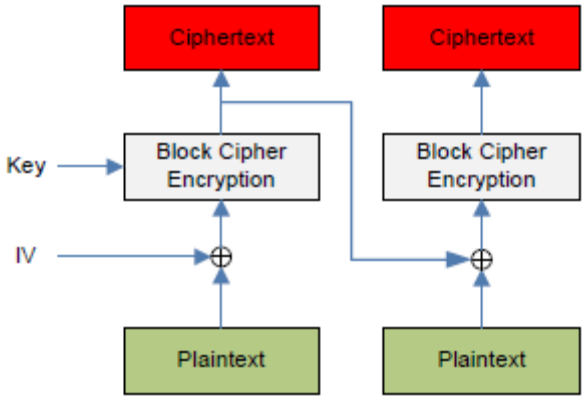

Figure 2. ECB block diagram

The following figure shows the drawback of the ECB mode. Taking the Freescale logo as an example it is still visible in the encoded form using this mode. It is obvious that this is not very secure.

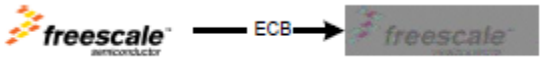

Figure 3. Encoding using ECB mode

3.2 Cipher-block chaining (CBC)

The Cipher-block (CBC) mode, invented in 1976, is one of the most important cipher modes. In this mode the output of the last encoding step is xor'ed with the input block of the actual encoding step. Because of this, an additional value for the first encoding step is necessary which is called initialization vector (IV). Using this method each cipher block depends on the plaintext blocks processed up to that point.

The following figure shows the block diagram of the CBC mode.

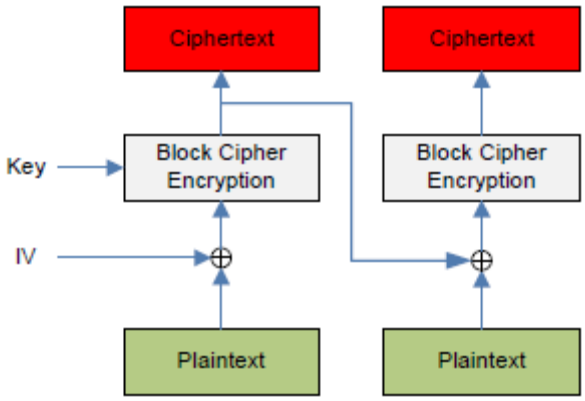

Figure 4. CBC block diagram

The following figure shows the encoding result of the Freescale logo using the CBC cipher mode. The difference from the ECB mode is self-evident. In many applications ECB mode may not be appropriate.

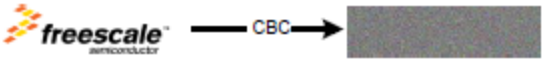

Figure 5. Encoding using CBC mode

3.3 CMAC (Cipher-based Message Authentication Code)

A CMAC provides a method for authenticating messages and data. The CMAC algorithm accepts as input a secret key and an arbitrary-length message to be authenticated, and outputs a CMAC. The CMAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any change in the message content

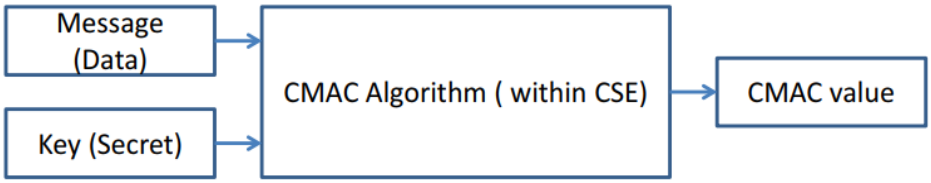

Figure 6. CMAC Scheme

If you want more information about CSE functional description and CSE commands, see MPC5777C reference manual.

4 CRYPTO module in MCAL4.3

The following figure shows the location of Crypto Driver module in the micro controller abstraction layer. It is below the Crypto Interface module and Crypto Service Manager module. It implements a generic interface for synchronous and asynchronous cryptographic primitives. It also supports key storage, key configuration, and key management for cryptographic services.

To provide cryptographic functionalities an ECU needs to integrate one unique Crypto Service Manager module and one Crypto Interface. However, the Crypto interface can access several Crypto Drivers, each of them is configured according to the underlying Crypto Driver Object.

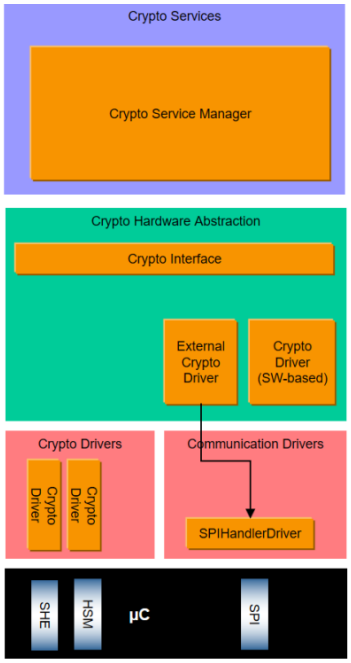

Figure 7. AUTOSAR layered view with crypto module

A Crypto Driver Object represents an instance of independent crypto hardware “device” (e.g. AES accelerator). There could be a channel for fast AES and CMAC calculations on a HSM for jobs with high priority, which ends on a native AES calculation service in the Crypto Driver. But it is also possible that a Crypto Driver Object is a piece of software, e.g. for RSA calculations where jobs are able to encrypt, decrypt, sign or verify. The Crypto Driver Object is the endpoint of a crypto channel.

NOTE

Crypto have layers including Crypto Cryif and CSM, since CSM is always a stub and only in order to avoid compiler error. The job_configuration_structure is responsible by CSM, so the job structure cannot generated by NXP CSM itself, as CSM is a stub in MCAL perspective. Developers need to manually update the structure and passing it to Crypto_Process_Job. So if need more CSM package support and should contact the third party(i.e vector DaVinci).

Figure 8 shows the relationship between different configuration items in EB:

Cryptoprimitives ->CryptoDriverObject->CryIfChannel->CsmQueue->CsmJobs

CryptokeyElement->CryptokeyType->Cryptokey->CryIfKey->CsmKeys

Crypto Driver Object: A Crypto Driver implements one or more Crypto Driver Objects. The Crypto Driver Object can offer different crypto primitives in hardware or software. The Crypto Driver Objects of one Crypto Driver are independent of each other. There is only one workspace for each Crypto Driver Object (i.e. only one crypto primitive can be performed at the same time)

CryptoKeyElement: Key elements are used to store data. This data can be key material or the IV needed for AES encryption. It can also be used to configure the behavior of the key management functions.

CryptoKeyType: A key type consists of references to key elements. The key types are typically pre-configured by the vendor of the Crypto Driver.

CryptoKey: A Key can be referenced by a job in the CSM. In the Crypto Driver, the key references a specific key type.

CryptoPrimitive: A crypto primitive is an instance of a configured cryptographic algorithm realized in a Crypto Driver Object.

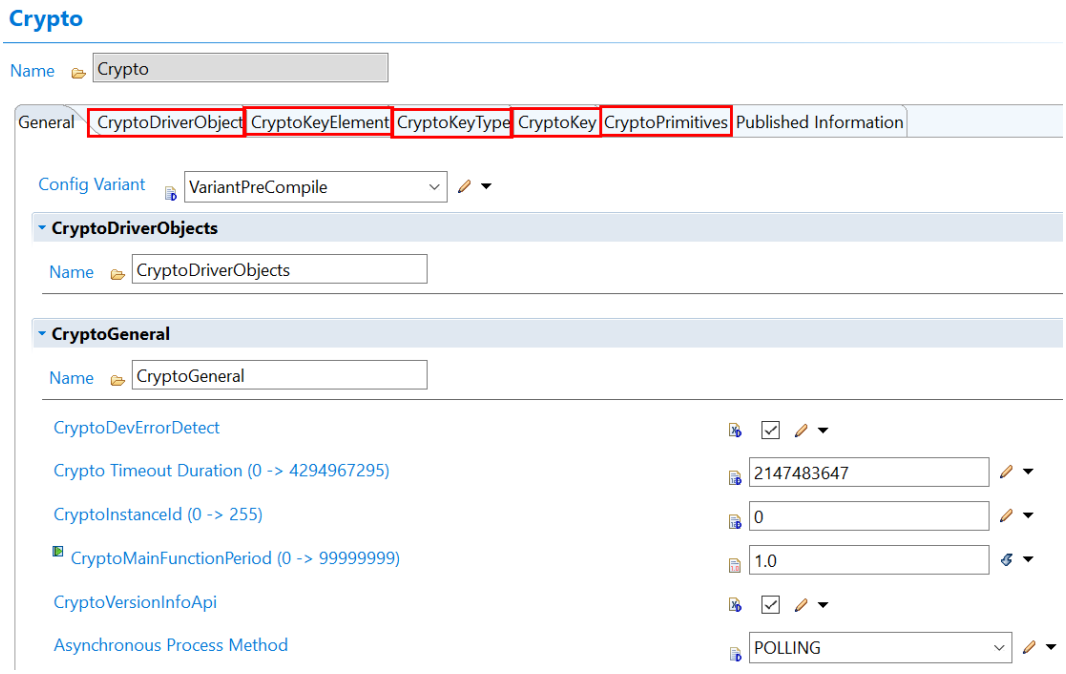

Figure 8. Crypto configuration in EB

CryIf: The crypto drivers are called by CryIf, the Crypto drivers access the underlying hardware and software objects to calculate results with their cryptographic primitives. The results are forwarded to CryIf.

CsmJob: A job is an instance of a job's configured in cryptographic primitive. An operation of a crypto primitive declares what part of the crypto primitive will be performed. There are three different operation modes:

- START is a operation mode indicates a new request of a crypto primitive and will be cancel all previous request of the same job and preemptive
- UPDATE mode indicates that the crypto primitive expects input data
- FINISH mode indicates that after this part all data are fed completely and the crypto primitive can finalize the calculation

The priority of a job defines the importance of it. The higher the priority means more immediately the job is executed. The priority of a cryptographic job is part of the configuration.

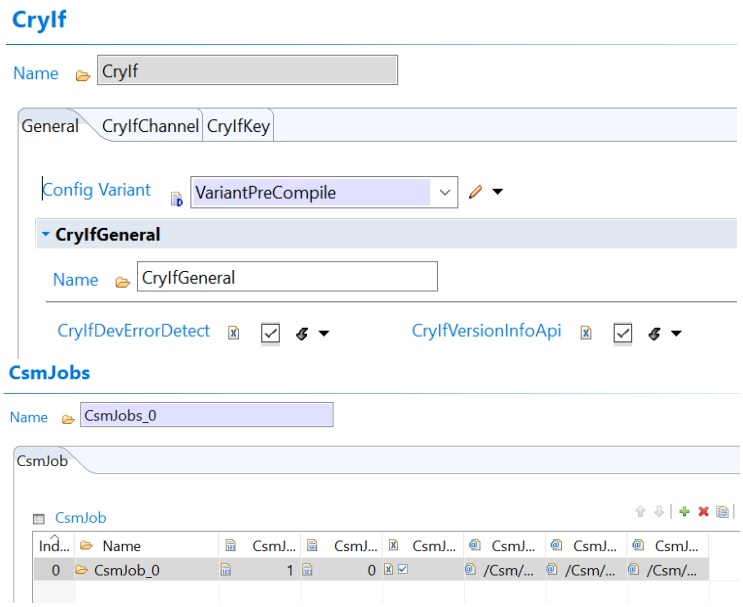

Figure 9. Cryif and CsmJobs in EB

NOTE

The crypro driver does not have callback function in CryIf.c file, so it should add SampleAppCrypto(job, result) into CryIf_CallbackNotification(const Crypto_JobType* job, Std_ReturnType result) function in CryIf.c file.

As show in the following figure, this sample configure three primitives, ENCRYPT, RNG(random number generated) and DECRYPT.

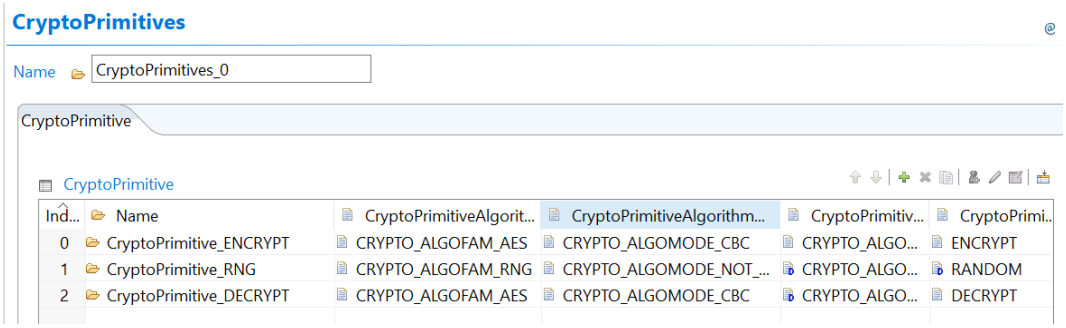

Figure 10. CryptoPrimitive configuration in EB

As show in the following figure, A CryptoKeyElement having the CryptoKeyElementId set to 1 represents a key material and cannot be set be using the field CryptoKeyElementInitValue. All the other CryptoKeyElementIds can be set either using CryptoKeyElementSet function or the Tresos field CryptoKeyElementInitValue.

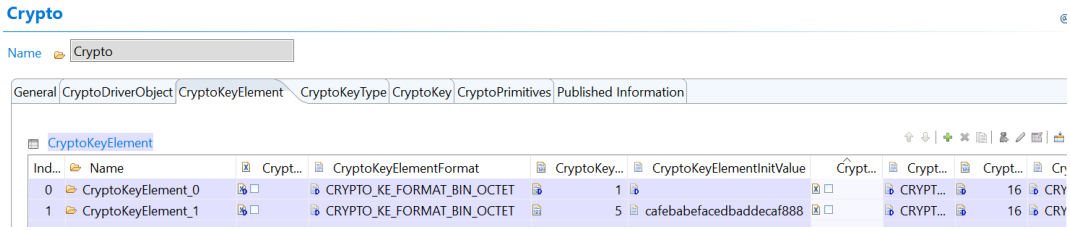

Figure 11. CryptoKeyEelment configuration in EB

As show in the following figure, key elements and keys have to be configured for all primitives supported in this release. Containers CryptoKeyElements, CryptoKeyTypes and CryptoKeys should be activated or deactivated in Tresos in the same time. For a key it is mandatory to have a key type and configured key elements. The index of the different key elements from the different Crypto services are defined as in imported types table SWS_Csm_01022(in AUOTOSAR document Specification of Crypto Service Manager)

A key has a state which is either 'valid' or 'invalid'. By default, all the keys are 'invalid' and have to be set to valid by using the function Crypto_KeySetValid. If a key is in the invalid state then the Crypto services which make use of the key returns CRYPTO_E_KEY_NOT_VALID value.

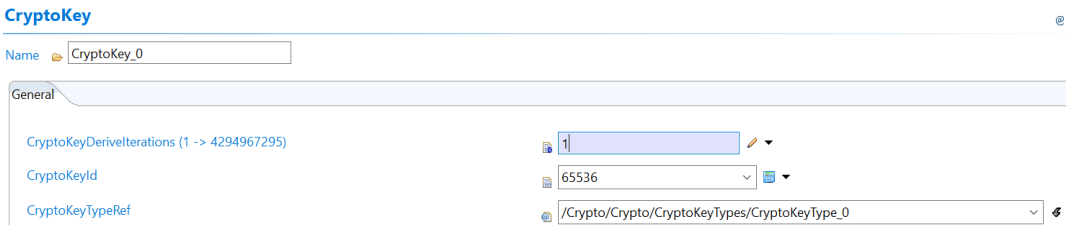

Figure 12. CryptoKey configuration in EB

Because crypto driver not include CSM layer, so the Crypto_JobType structure should be initialized manually in the code.

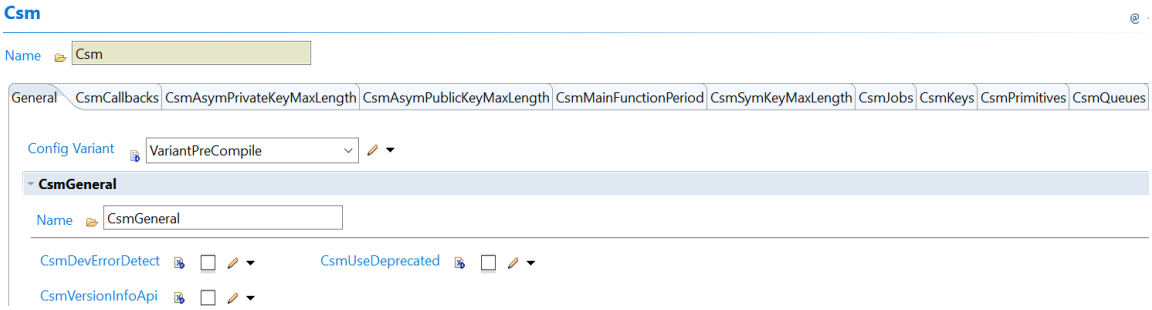

Figure 13. Csm in EB

5 CRYPTO loading key and processing primitive

To process a primitive (random number generation, MAC generation or verification, AES encrypt/decrypt), the following sequence should be followed:

1. If keys are needed, the containers CryptoKeyElements, CryptoKeyTypes, CryptoKeys should be enabled
2. Crypto_KeyElementSet(65536, CryptoKeyElementId_0, aes_test01_key, 16) meaning a key material corresponding to key 65536 and having the size 16 bytes is configured
3. Call the API function Crypto_KeySetValid(65536) to enable key 65536
4. Call the API function Crypto_ProcessJob() to process job, it process three jobs(random generated, encryption and decryption) in this sample code

```
/*Sample set three jobs: random generated, encryption and decryption*/
case CRYPTO_JOB_START:
  /*Job 1 to process random generate*/
 stdRet = Crypto_ProcessJob(CryptoDriverObjectId, &Crypto_JobType_info[0]);
   if (stdRet != E_OK)
     {
         pstSampleAppData->stCryptoData.ucState = CRYPTO_ERROR_STATE;;
         return (E_NOT_OK);
     }
    /*Job 2 to process Encryption*/
 stdRet = Crypto_ProcessJob(CryptoDriverObjectId, &Crypto_JobType_info[1]);
   if (stdRet != E_OK)
     {
         pstSampleAppData->stCryptoData.ucState = CRYPTO_ERROR_STATE;
         return (E_NOT_OK);
     }

   /*Job 3 to process Decryption*/
 stdRet = Crypto_ProcessJob(CryptoDriverObjectId, &Crypto_JobType_info[2]);
   if (stdRet != E_OK)
     {
         pstSampleAppData->stCryptoData.ucState = CRYPTO_ERROR_STATE;
         return (E_NOT_OK);
     }
         pstSampleAppData->stCryptoData.ucState = CRYPTO_JOBPROCESS_STATE;
         break;
```

Figure 14. Process job in sample code

Call API function StringCompare ((uint8_t*)ucPlainText, ucDecText, 16) to verify the encryption and decryption functionality.

```
stdRet = StringCompare((uint8_t*)ucPlainText, ucDecText, 16);
  if (stdRet != E_OK)
    {
        pstSampleAppData->stCryptoData.ucState = CRYPTO_ERROR_STATE;
        return (E_NOT_OK);
    }
        pstSampleAppData->stCryptoData.ucState = CRYPTO_COMPLETE_STATE;
        break;
```

Figure 15. Compare the ucPlainText and ucDecText

6 References

- MPC5777C Reference Manual (Document ID: MPC5777CPRM)
- Specification of Crypto Service Manager(Document link)
- Specification of Crypto Driver(Document link)
- AUTOSAR_MCAL_CRYPTO_IM
- AUTOSAR_MCAL_CRYPTO_UM

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN 13061
Rev. 0
12/2020