DORCA3 API

Contents

[1 SPI MODE 3](#_Toc525817563)

[2 RSA 3](#_Toc525817564)

[2.1 Definition of Module 3](#_Toc525817565)

[2.2 Block Diagram 3](#_Toc525817566)

[3 AES/ARIA 5](#_Toc525817567)

[3.1 Module Definition 5](#_Toc525817568)

[3.2 Block Diagram 5](#_Toc525817569)

[3.3 API 5](#_Toc525817570)

[3.4 Block Cipher Modes of Operation 7](#_Toc525817571)

[4 Cipher Memory 13](#_Toc525817572)

[4.1 모듈의 정의 13](#_Toc525817573)

[4.2 블록도 설명 13](#_Toc525817574)

[4.3 API 13](#_Toc525817575)

[5 ECDH 15](#_Toc525817576)

[5.1 Module Definition 15](#_Toc525817577)

[5.2 Block Diagram 15](#_Toc525817578)

[5.3 API 16](#_Toc525817579)

[6 PUF&TRNG 17](#_Toc525817580)

[6.1 Module Definition 17](#_Toc525817581)

[6.2 블록도 설명 17](#_Toc525817582)

[6.3 API 17](#_Toc525817583)

[7 SHA 18](#_Toc525817584)

[7.1 Module Definition 18](#_Toc525817585)

[7.2 블록도 설명 18](#_Toc525817586)

[7.3 API 18](#_Toc525817587)

[8 ECDSA 19](#_Toc525817588)

[8.1 Module Definition 19](#_Toc525817589)

[8.2 Block Diagram 19](#_Toc525817590)

[8.3 API 19](#_Toc525817591)

[9 UUP: User Unique Password 21](#_Toc525817592)

[9.1 21](#_Toc525817593)

[9.1 Module Definition 21](#_Toc525817594)

[9.2 Block Diagram 21](#_Toc525817595)

[9.3 API 21](#_Toc525817596)

[10 Key Generation and Modification 22](#_Toc525817597)

[10.1 Module Definition 22](#_Toc525817598)

[10.2 Block Diagram 22](#_Toc525817599)

[10.3 API 22](#_Toc525817600)

[11 Setup procedure 23](#_Toc525817601)

[12 Permission 23](#_Toc525817602)

[12.1 Module Definition 23](#_Toc525817603)

[12.2 Block Diagram 23](#_Toc525817604)

[12.3 API 23](#_Toc525817605)

[13 Configuration Domain 24](#_Toc525817606)

[13.1 Module Definition 24](#_Toc525817607)

[13.2 Block Diagram 24](#_Toc525817608)

Reference

SPI comm. definition

<https://en.wikipedia.org/wiki/Serial_Peripheral_Interface>

1. SPI MODE

CORE 1 manages the asymmetric code, and CORE 0 manages the symmetric code. The SPI\_MODE needs to be set as follows:

**CORE 0: Symmetric Key, Cipher memory, AES/ARIA etc.**

CPOL = 0, CPHA = 0, Clock idle low, data is clocked in on rising edge, output data (change) on falling edge

**CORE 1: Asymmetric Key, ECDSA, RSA, ECDH etc.**

CPOL = 0, CPHA = 1, Clock idle low, data is clocked in on falling edge, output data (change) on rising edge

Reference

Public-key cryptography

<https://en.wikipedia.org/wiki/Public-key_cryptography>

Symmetric vs. Asymmetric Encryption – What are differences?

https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

RSA (cryptosystem)

<https://en.wikipedia.org/wiki/RSA_(cryptosystem)>

1. RSA
   1. Definition of Module

RSA 2048 Encryption and Decryption

* 1. Block Diagram



The private key decodes the text encoded with the public key.

API

**Module Name: RsaPubEnc2048**

**Description: performs RSA encryption**

**Parameter:**

* unsigned char \* pub\_key\_n [in] RSA Modulus
* unsigned char \* pub\_key\_e [in] RSA PublicExponent
* unsigned char \* out [out] output data
* unsigned char \*in [in] input data
* size\_t len [in] data length set to 512
* int padding [in] not used

**Function Name: RsaPubDec2048**

**Description: performs RSA Decryption**

**Parameter:**

* unsigned char \* priv\_key [in] RSA Private key
* unsigned char \* pub\_key\_n [in] RSA Modulus\_n
* unsigned char \* out [out] output data
* unsigned char \*in [in] input data
* size\_t len [in] data length set to 512
* int padding [in] not used

Reference

Symmetric-key algorithm

<https://en.wikipedia.org/wiki/Symmetric-key_algorithm>

1. AES/ARIA
   1. Module Definition

AES128/AES256/ ARIA128/ ARIA256

* 1. Block Diagram



The keys used for encoding and decoding are the same using the symmetric key algorithm

* 1. API

**Module Name: Dorca3CipherDecipher**

**Description: Performs AES encryption and decryption**

**Parameter:**

* int mode [in] 0: encryption 1: decryption
* int arg\_type [in] 1: AES 2: ARIA
* unsigned char\* key [in] key for encryption and decryption

If the key value is NULL, maintain the original key value

* int key\_length [in] key length
* unsigned char\* iv [in] Initialization Vector
* unsigned char\* out [out] output buffer
* unsigned char\* in [in] input buffer
* size\_t len [in] length of buffer 16 or 32
* int type [in] operation mode type {

MODE\_ECB = 0,

MODE\_CBC = 1,

MODE\_OFB = 2,

MODE\_CTR = 3,

MODE\_CFB = 4

* int LAST [in] indicator to end of sequence 0: on going 1: end

**Module Name: Dorca3CipherDecipherGivenIdx**

**Description: Performs encryption and decryption with key value set previously. Key is saved once encryption and decryption have taken place.**

**Parameter:**

* int key\_idx [in] key index
* int mode [in] 0: encryption 1: decryption
* int arg\_type [in] 1: AES 2: ARIA
* unsigned char\* key [in] not used
* int key\_length [in] key length
* unsigned char\* iv [in] Initialization Vector
* unsigned char\* out [out] output buffer
* unsigned char\* in [in] input buffer
* size\_t len [in] length of buffer 16 or 32
* int type [in] operation mode type {

MODE\_ECB = 0,

MODE\_CBC = 1,

MODE\_OFB = 2,

MODE\_CTR = 3,

MODE\_CFB = 4

* int LAST [in] indicator to end of sequence 0: on going 1: end

Reference

Advanced Encryption Standard

https://en.wikipedia.org/wiki/Advanced\_Encryption\_Standard

Block cipher mode of operation

https://en.wikipedia.org/wiki/Block\_cipher\_mode\_of\_operation

3. 4. Block Cipher Modes of Operation

In the study of cryptography, block cipher modes of operation is the process of operating under block cipher repetitively and safely using a single key. Because block cipher operates under specific length, it is important to first divide any length of data into unit blocks and then to decide how to encode them. This is when the cipher modes are determined by the modes of operation.

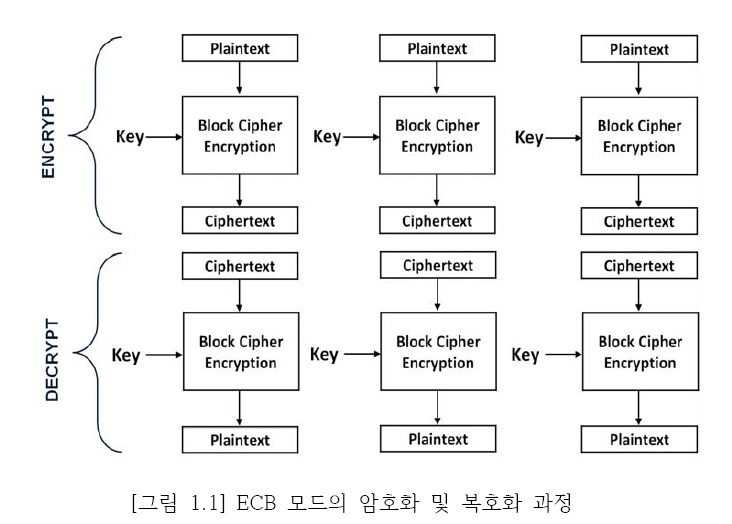
**Modes of Operation**

To improve the weakness in security of ECB, the basic method of block cipher, different modes of operation are used. Block cipher modes of operation utilize Initialization Vector (IV) and the results of encryption and decryption of the previous block to cause effect on the encryption and decryption of the present block, therefore enhancing the security of the entire process. The most well-known modes include ECB, CBC, OFB, CFB, and CTR, and their reliability and transmission rate between the blocks are different.

1. ECB (Electronic Code Book) Mode

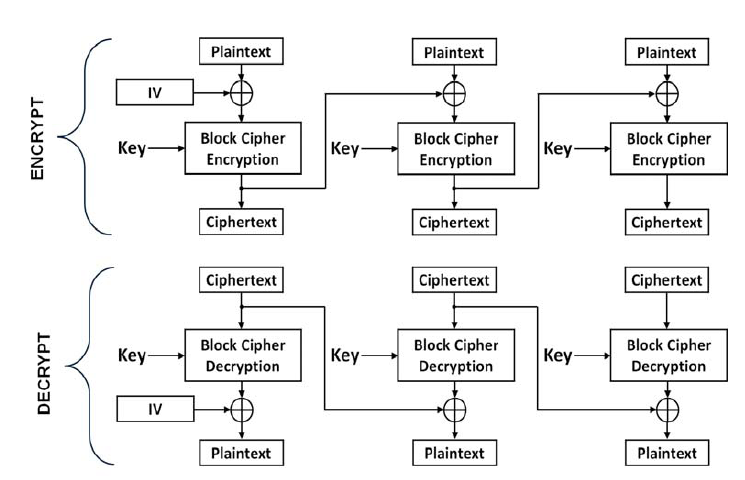
ECB As it could be seen in the figure below (fig. 1.1), the encryption of the ECB mode encrypts each block of data independently. If the length of the snippet of the data isn’t a factor of the length of the data, the snippet will undergo padding to obtain the desired length. The keys used during encryption and decryption are the same. After the key value has been set, 물어보고

Because each block is independently encrypted, an error in one block won’t affect the operation of another block.



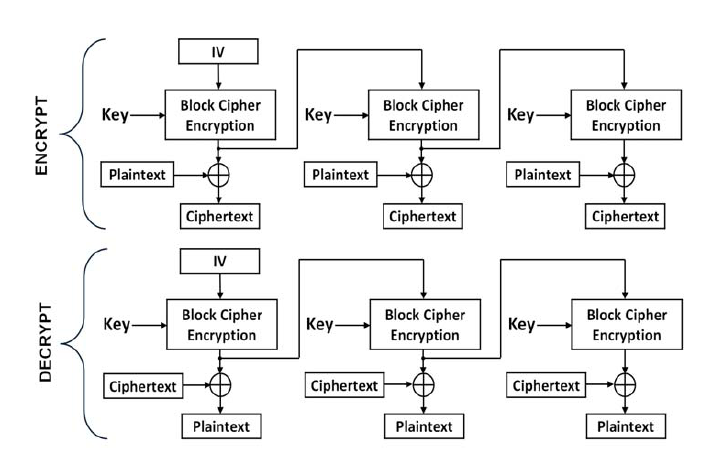
1. CBC (Cipher Block Chaining) Mode

As it could be seen in the figure below (fig. 1.2), the encryption of the CBC mode ciphers each block consecutively depending on the encryption of the previous block which is done with XOR. In the first block, the primitive vector (IV) is utilized. Because encryption requires feedback from the previous block, it is impossible to cipher the entire data parallelly, but in the case of decryption, the results from the previous block are XORed after deciphering has completed, parallel computation is possible. CBC mode has high security thanks to its reliance on the result of the previous block, which means that many different versions of ciphers could be produced from a single piece of data. Therefore, this mode is used mostly when handling personal or confidential information.



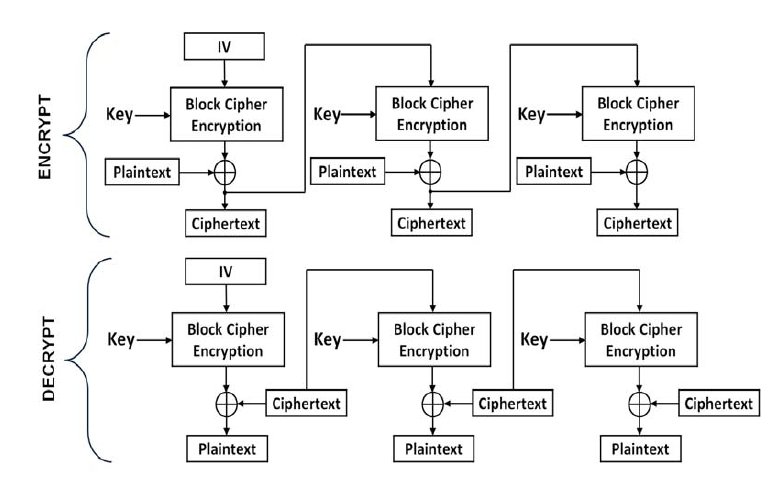
1. OFB (Output Feedback) Mode

As it could be seen in the figure below (fig. 1.3), the encryption of the OFB mode cyphers the plaintext by XORing it with block cypher encryption, which outputs the ciphertext. As for the first block, the initialization is done with the primitive vector, and for the succeeding blocks, the cipher block sends signals to do so. In terms of processing, it is the same for both encryption and decryption.



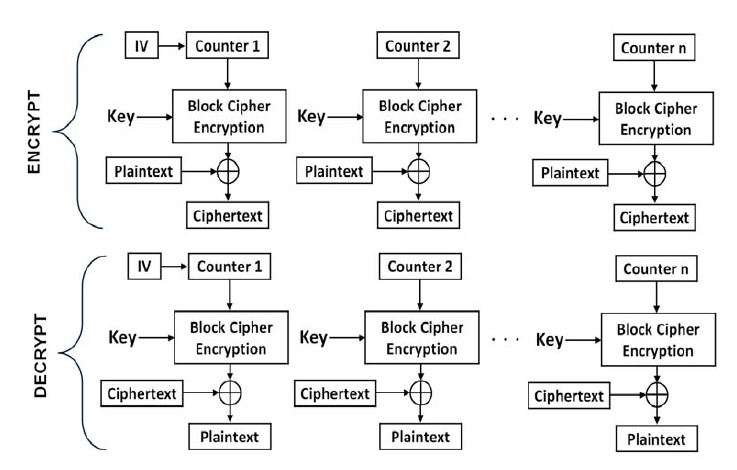
1. CFB (Cipher Feedback) Mode

As it could be seen in the figure below (figure 1.4), the encryption of the CFB mode is done the same as the CBC mode and the ciphertext is obtained by XORing the plaintext and the encryption in the given block. The decryption is performed by XORing the previous ciphertext with the block encryption and therefore producing the plaintext. For both encryption and decryption, the first block encrypts the primitive text, which is XORed with the plaintext(ciphertext), which outputs a ciphertext(plaintext). In both encryption and decryption procedure, only the encryption of the block cipher is used, and as in the case of CBC mode, encryption is a serial process while decryption could be performed in a parallel manner.



1. CTR (counter) Mode

As it could be seen in the figure below (fig. 1.5), the encryption of the CTR mode increments the value of initial vector by 1, which is XORed with the ciphertext with the plaintext to produce the ciphertext. Encryption and decryption process are performed in the same manner, and cryptic algorithm is used. The process don’t rely on one another, therefore it could be processed parallelly which makes it suitable for high-speed encryption/decryption.



1. Cipher Memory
   1. Module Definition

3840 byte cipher memory

* 1. Block Diagram



This module is used to save or read from non-volatile memory eeprom, and when the user attempts to send data with a designated key, DORCA3 ciphers and saves the data with that key. When EepromWrite is used to write a Cipher text the Dorca3 ciphers it and saves it. When EepromRead is used to read data, Dorca3 again ciphers it and returns the data.

Eeprom could be distinguished as Page, Sub Page, and Sub Frame as shown in the chart below.

|  |  |  |
| --- | --- | --- |
|  | SUBFRAME | 3'h0 : [127:0] of selected page 3'h1 : [255:128] of selected page 3'h2 : [383:256] of selected page 3'h3 : [511:384] of selected page |
|  | SUBPAGE | 3'h0 : subpage 0 (0x00 ~ 0x3F) 3'h1 : subpage 1 (0x40 ~ 0x7F) 3'h2 : subpage 2 (0x80 ~ 0xBF) 3'h3 : subpage 3 (0xC0 ~ 0xFF) |
|  | PAGE | 4'h1 : EE\_USER\_ZONE\_M01 (0xF100 ~ 0xF1FF) … 4'hF : EE\_USER\_ZONE\_M15 (0xFF00 ~ 0xFFFF) |

* 1. API

**Function name: EepromWrite**

**Description: write encrypted data into eeprom using a designated key**

**Parameter:**

* Unsigned char \* Key[in] Key value used for encryption
* Setting key value to NULL keeps the original value of the key.
* int SubFrame[in] SubFramenumber
* int SubPage[in] SubPageNumber
* int Page,[in] PageNumber
* int Algorithm[in] AES128 3,AES256 1,ARIA128 2,ARIA256 0
* unsigned char \*CT[in] Ciphertext to be saved

**Function Name: EepromRead**

**Description: Outputs a common key from private and public keys inputed**

**Parameter:**

* Unsigned char \* Key[in] Key value that will be used for decryption
* Setting key value to NULL keeps the original value of the key.
* int SubFrame[in] SubFramenumber
* int SubPage[in] SubPageNumber
* int Page,[in] PageNumber
* int Algorithm[in] AES128 3,AES256 1,ARIA128 2,ARIA256 0
* unsigned char \*CT[out] Ciphertext to be read

Reference

ECC : Elliptic-curve cryptography

https://en.wikipedia.org/wiki/Elliptic-curve\_cryptography

Elliptic-curve Diffie–Hellman

https://en.wikipedia.org/wiki/Elliptic-curve\_Diffie%E2%80%93Hellman

1. ECDH
   1. Module Definition

ECDH P256

* 1. Block Diagram



This module is used to create secure and common keys between two users. As it could be seen in the diagram above, USER A creates his/her own private key DA, and by perforing DA \* G, HA is created which is passed to USER B. USER B also creates his/her own pivate key DB which will create HB by doing DB \* G, and pass that to USER A. Finally, both USER A and USER B creates a common key called S by S = DAHB = DADBG for USER A and S = DBHA = DADBG for USER B.

* 1. API

**Function name: EcdhGenPubKey**

**Description: Receives public key by inputting secure key.**

**Parameter:**

uint8\_t\* sk[in] Private Key length: 32

struct \_point\* p1[out] Public KEy struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

**Function Name: EcdhGenPubKeyPuf**

**Description: Generates a random value, makes it into a secure key, and uses it to create a public key.**

**Parameter:**

struct \_point\* p1[out] Public Key struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

**Function Name: EcdhGenSessionKey**

**Description: Creates the final common key using the secure key from input.**

**Parameter:**

uint8\_t\* sk[in] 비밀키 length: 32

struct \_point\* p1[out] 공개키 struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

**Function Name: EcdhGenSessionKeyPuf**

**Description: Uses the secret key that was created before to create the final final common key. Ecdh\_gen\_pub\_key\_puff must be called before to create the secure key.**

**Parameter:**

struct \_point\* p1[out] 공개키 struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

**Function Name: KeyFromEcdh**

**Description: Places the key obtained from ECDH into AES\_X0**

**Parameter:**

None

Reference

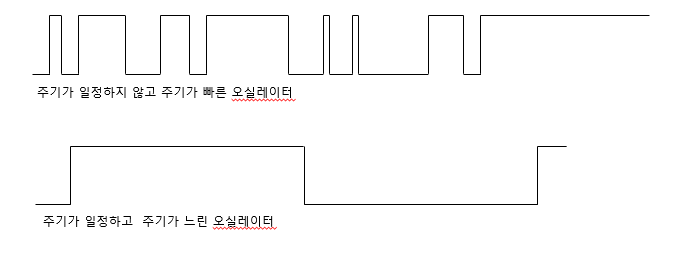
TRNG making:

<https://www.google.co.kr/search?q=TRNG+AES+feedback+LFSR&safe=active&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi2qpjUiNrdAhUBebwKHeBtDPkQ_AUICigB&biw=1466&bih=817#imgrc=U4iYFEIm2O1QVM>:

1. PUF&TRNG
   1. Module Definition

Generates random value.

* 1. 블록도 설명



By using an oscillator which has a low and even frequency to count the frequency of an oscillator which has high and uneven frequency, different counts could be obtained every time the count is performed. Using this, random number could be generated.

* 1. API

**Function name: rand\_pseudorand**

**Description: random number is generated**

**Parameter:**

* char \*out Output 16 byte

Reference

Hash

<https://en.wikipedia.org/wiki/Hash>

SHA-2

https://en.wikipedia.org/wiki/SHA-2

1. SHA
   1. Module Definition

Creates HASH table

* 1. 블록도 설명
  2. API

**Function name: StandardShaMode**

**Description: creates HASH values**

**Parameter:**

* unsigned char \*txdata [in]
* unsigned char \*rxdata [out]
* long long ByteNo [in]

Reference

Elliptic Curve Digital Signature Algorithm

<https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm>

1. ECDSA



6. 1. Module Definition

Elliptic Curve Cryptography

* 1. Block Diagram



ECDSA is an algorithm that sends message with digital cryptography, and when it receives the message with digital cryptography, it checks for any errors and/or illegal changes.

사용자 A는 Message를 HASH(SHA256)을 수행하여 digest를 만들고 이를 비밀키로 서명(Sign)을 하여 서명 데이터 R,S를 만든다

사용자 B는 수신한 Message를 HASH(SHA256)을 수행하여 digest를 만들고 수신한 R,S를 이용하여 검증(Verify)을 하여 Message의 위변조 여부를 확인한다

* 1. API

Function Name: EcdsaGenSignature

Description: generates digital signature

Parameter:

* uint8\_t\* d length: 32 secret key [in]
* uint8\_t\* k length: 32 secret key [in]
* uint8\_t\* r length: 32 sign [out]
* uint8\_t\* s length: 32 sign [out]
* uint8\_t\* h length: 32 sha-256 hash[in]

**Function Name: EcdsaVerifySignature**

**Description: verifies the digital signature**

**Parameter:**

* point \*public\_key typedef struct \_point

{

uint8\_t x[32];

uint8\_t y[32];

}point;

* uint8\_t\* r length: 32 sign [in]
* uint8\_t\* s length: 32 sign [in]
* uint8\_t\* h length: 32 sha-256 hash[in]

**Function Name: EcdsaGenPublicKey**

**Description: Creates Public Key**

**Parameter:**

* unsigned char \*private\_key length:32 Private Key[out]
* point \*public\_key typedef struct \_point

{

uint8\_t x[32];

uint8\_t y[32];

}point;

* uint8\_t\* r length: 32 sign [in]
* uint8\_t\* s length: 32 sign [in]
* uint8\_t\* h length: 32 sha-256 hash[in]

1. UUP: User Unique Password

   2. Module Definition

Description of User Unique Password, which is the only method of distinguishing one chip from another. This module was originally called Root Serial, but due to possible misinterpretation of the name which could lead to possible concerns regarding backdooring, the name has been changed to user unique password. In fact, this module does create a unique password that the user sets.

* 1. Block Diagram

Saves 4 Serials, and when reading a Serial, reads the result from SHA.

Once UID permission is obtained, the first and third serials can be obtained.

* 1. API

**Function Name: RSCreate01**

**Description: Algorithm of NeoWine that tags each device its own serial 0,1.**

**Parameter:**

None

**Function Name: RSCreate23**

**Description: With a value that user inputs, serial 2, 3 is created.**

**Parameter:**

* unsigned char \*pUservalue32A 32bit user value for serial2[in]
* unsigned char \*pUservalue32B 32bit user value for serial3[in]

**Function Name: RSSSHAReadIdx**

**Description: reads serial by performing SHA**

**Parameter:**

* Int index : 읽어올 serial index (0,1,2,3)
* Unsigned char\* out: serial 값

**Function name: RSDirectRead13**

**Description: Reads the actual serial number 1 and 3 by inputting UID and PASSWORD**

**Parameter:**

* unsigned char \*pRS: output [out]
* int index: index값 1또는 3 [in]
* unsigned char \*UID\_PW\_PT: 16바이트 UID 비밀번호 포인터[in]

1. Key Generation and Modification
   1. Module Definition

Generates or mFasodifies the key value that is used for encryption and decryption

* 1. Block Diagram



The key hierarchy is as follows: SEED\_KEY, AES\_KEY\_0, AES\_KEY\_1, AES\_KEY\_2, AES\_KEY\_3. SEED\_KEY is written when it is first created, with it AES\_KEY0 is made. With AES\_KEY1, AES\_KEY2 is created, and with AES\_KEY 2 AES\_KEY3 is created.

In the sample program, AES\_KEY\_1 is used for permission get operation, and AES\_KEY0 is designated for encryption and decryption

* 1. API

In the sample program, AES\_KEY\_1 is used for permission get operation, and AES\_KEY0 is designated for encryption and decryption

**Function name: KeyCreatXn**

**Description: Generates and modifies key**

**Parameter:**

* int index: Key index[in]
* int mode: MODE256 or MODE128[in]
* unsigned char \*prevKey: Key value of the key from previous hierarchy[in]
* unsigned char \*key: Key value previously in]

1. Setup procedure



The Seed Key is written during the initiation sage. After that, the Seed Key is used to make the AES KEY. ROOT SERIAL is created after AES KEY is made. When the download file is executed, writing is executed.

1. Permission
   1. Module Definition

With permission, the user is capable of performing a specific function and reading data.

Permission is comprised of SUPER, DETOUR, DESTROY0, DESTROY1, EEPROM, and UID.

* 1. Block Diagram

Omitted

* 1. API

**Function Name: GetPermission**

**Description: Receives permission**

**Parameter:**

* int iType [in]

RG\_PERM\_SUPER\_PASS = 5,

RG\_PERM\_DETOUR\_PASS = 4,

RG\_PERM\_DESTORY0\_PASS = 3,

RG\_PERM\_DESTORY1\_PASS= 2,

RG\_PERM\_EEPROM\_PASS= 1,

RG\_PERM\_UID\_PASS = 0

* unsigned char \*pchPW Password for Permission[in]
* int index Key used to obtain permission index[in]

**Function Name: ReleasePermission**

**Description: Releases permission**

**Parameter:**

none

1. Configuration Domain
   1. Module Definition

Domain used to set up the chip. More explanation later.

* 1. Block Diagram

Configuration memory could be distinguished as the following types. Each type has following characteristics.

-EE\_CONFIC\_NW

From the package, Neowine decides a value to write for sale. The value set by Neowine operates within the device. External MCU can’t read nor write this value, and if necessary, external MCU software library needs to be used. With the permission of EE\_SUPER\_PW, the value can be manually modified by API.

-EE\_CONFIC\_FAC

From the package, Neowine decides a value to write for sale. This value set by Neowine operates within the device. External MCU can read this value using BlockRead command as plain text. With permission of EE\_SUPER\_PW, the value can be manually modified by API.

-EE\_CONFIC\_UID

With the permission of EE\_UID\_PW, it can be modified with API.

-EE\_SEED\_KEY

Refer to the description of Key Memory.

With permission of EE\_SUPER\_PW, it can be modified with API.

-EE\_CONFIG\_LOCK

With permission of EE\_SUPER\_PW, it can be modified with API.

EE\_UZx0\_WR\_LOCK ~ EE\_UZxF\_WR\_LOCK

EE\_SEED\_KEY\_LOCK

EE\_KEY\_AES\_LOCK

|  |  |  |  |
| --- | --- | --- | --- |
| EE\_CONFIG\_NW  0xEB40 |  |  | EE\_CONFIG\_NW  Neowine sets the value. |
| EE\_GID | 4 | Lowest Byte Address is EE\_GID[0] byte. …. Highest Byte Address is EE\_GID[3] byte. Default: EE\_GID[3],[2],[1],[0] = 0x2222\_1111 |
| EE\_CONFIG\_NW\_CTRL0 | 1 | [0] : EE\_OKA\_10\_11N  0 : AES-128 communication 1:1, 1:N and 1:0 possible  However, receives control of RG\_AES\_10\_11N.  1 : 1:0 communication  0 : 1:1 or 1:N communication (default)  1 : AES-128 communication 1:0 only possible  [1] : EE\_OKA\_OVERRIDE, OKA0\_E, OKA2\_D 단계에서 MCU가 AES-128 key override 가능 여부 제어.  0 : MCU may not randomly input key value. (default)  1 : MCU may randomly input key를 value.  [2] : EE\_TWIST\_USE (NOT USED)  0 : Twist is not performed during AES encryption and decryption.  1 : Twist is performed during AES encryption and decryption.  [3] : EE\_TWIST\_DORCA30 (NOT USED)  0 : Compatible with DORCA20.  1 : Twist mode newly implemented in DORCA30. Not compatible with DORCA20  [4] : EE\_STANDBYN  Managed by E-MCU. Active low signal which indicates the Standby state. Controls the ON/OFF state of the main power supply  0 : Standby Phase. Analog’s main power supply is turned OFF  1 : Not in Standby phase. Analog’s main power supply is turned on  When in Standby phase, the amp power supplier is always ON.  [7:5] RESERVED  0x0F |
| EE\_CONFIG\_NW\_CTRL1 | 1 | [0] : EE\_DETOUR\_ALLOW  b1 : Back door allow  b0 : Back door do not allow (default)  [1] : EE\_ARIA\_DISABLE, Used to disable DORCA-30. (NOT USED)  0 : ARIA algorithm is available  1 : ARIA algorithm is unavailable  [7:4] RESERVED  0x03 |
|  |  |  |
| EE\_CONFIG\_FAC  0xEB80 |  |  | EE\_CONFIG\_FAC  Neowine sets the value before sale.  E-MCU가 write 할 때는 EE\_\_SUPER\_PW permission 필요하고, E-MCU가 ST0\_EE\_CFG state에서 읽을 때는 permission 이 필요 없다. |
| EE\_LOT\_CODE | 8 | Managed and written by Neowine before sale. User may only read from it. 0x4444\_3333\_2222\_1111 |
| EE\_FACTORY\_CODE | 2 | Managed and written by Neowine before sale. User may only read from it. 0x5555 |
|  |  |  |  |
| |  | | --- | | EE\_CONFIG\_UID  0xEBC0 | | | | | |  |  | EE\_CONFIG\_UID Value set by user input.  If in lock status by EE\_CONFIG\_LOCK:EE\_CONFIG\_UID\_LOCK, write is restricted. |
| EE\_UID | 4 | Lowest Byte Address is EE\_UID[0] byte. …. Highest Byte Address is EE\_UID[3] byte. 0x4444\_4444 |
| EE\_MANUFACTID | 2 | EE\_MANUFACTID[0] EE\_MANUFACTID[1] 0x2222 |
| EE\_REV\_CTRL\_USR | 4 | Reserved Revision Control for user use. Limited access to EE\_UID\_PW.  0x4444\_4444 |
| EE\_CONFIG\_USER\_CTRL1 | 1 | [7:2] Reserved [1] EE\_CLKXTAL\_EN 0 : CLKXTAL can toggle high and low. (default) 1 : CLKXTAL maintains high state.  [0] EE\_OSC\_5M\_20M 0 : 20 MHz (default) 1 : 5 MHz |
| EE\_SEED  0xEC00 | EE\_SEED\_KEY | 32 | User(E-MCU) can receive UID\_PW permission to read and write from this domain in administrative mode(EE\_CONFIG\_LOCK:EE\_SEED\_KEY\_LOCK is in unlock status).  If EE\_CONFIG\_LOCK:EE\_SEED\_KEY\_LOCK is in Lock status, the user(E-MCU) may not read nor write EE\_SEED\_KEY even if the user has the UID\_PW permission. 0x0001\_0203\_0405\_0607\_0809\_0A0B\_0C0D\_0E0F\_1011\_1213\_1415\_1617\_1819\_1A1B\_1C1D\_1E1F |
| EE\_CONFIG\_USER  0xEC40 | EE\_CONFIG\_USER\_CTRL0 | 1 | In ST0\_EE\_CFG state, E-MCU can’t write anything in this page In ST0\_EE\_CFG state, E-MCU may freely read from this page.  [7:3] Reserved [2] EE\_RS\_x2\_FLAG  0 : Before EE\_RS\_x2 is made 1 : EE\_RX\_x2 is made [1:0] :RESERVED |
| |  | | --- | | EE\_CONFIG\_LOCK  0xEC80 | | | | | EE\_UZx0\_WR\_LOCK ~  EE\_UZxF\_WR\_LOCK | 16 | Setting bit (0->1) is only possible with EE\_UID\_PW permission. It is a OTP(One Time Programmable) Domain. The user (E-MCU) may change the value 0 from each bit to 1, but changing the value 1 to 0 is impossible. EE\_USER\_ZONE\_M00 ~ EE\_USER\_ZONE\_M15 Out of 16 domains, matched to 1  EE\_UZx0\_WR\_LOCK -> EE\_USER\_ZONE\_M11 matched to 1 Write is prohibited when status is Lock. 0x00 : EE\_UZxN domain is in Unlock status. 0x01~0xFF : EE\_UZxN domain is in Lock status. Changes can’t be made. |
| EE\_SEED\_KEY\_LOCK | 1 | Setting bit (0->1) is only possible with EE\_UID\_PW permission. It is a OTP(One Time Programmable) Domain. The user (E-MCU) may change the value 0 from each bit to 1, but changing the value 1 to 0 is impossible.  0x00 : EE\_SEED\_KEY domain is in Unlock status.  0x01~0xFF : EE\_SEED\_KEY domain is in Lock status. No changes can be made. |
| EE\_RS\_x0 LOCK | 1 | Setting bit (0->1) is only possible with EE\_UID\_PW permission. It is a OTP(One Time Programmable) Domain. The user (E-MCU) may change the value 0 from each bit to 1, but changing the value 1 to 0 is impossible.  0x00 : EE\_RS\_x0 domain is in Unlock status. E-MCU may read the values in EE\_RS\_x0 domain. 0x01~0xFF : EE\_RS\_x0 domain is in Lock status. E-MCU may not read the values in EE\_RS\_x0 domain. |
| EE\_KEY\_AES\_LOCK | 1 | Setting bit (0->1) is only possible with EE\_UID\_PW permission. It is a OTP(One Time Programmable) Domain.  The user (E-MCU) may change the value 0 from each bit to 1, but changing the value 1 to 0 is impossible. 0x00 : EE\_KEY\_AES\_xN domain is in Unlock status. E-MCU may read the values in EE\_KEY\_AES\_xN domain. 0x01~0xFF : EE\_KEY\_AES\_xN domain is in Lock status. E-MCU may not read the values in EE\_AES\_xN domain. |

13.3 API

Function Name: SetConfigArea

Parameter:

* int permType Permission type [in]

G\_PERM\_SUPER\_PASS = 5,

RG\_PERM\_DETOUR\_PASS = 4,

RG\_PERM\_DESTORY0\_PASS = 3,

RG\_PERM\_DESTORY1\_PASS= 2,

RG\_PERM\_EEPROM\_PASS= 1,

RG\_PERM\_UID\_PASS = 0

* int CfgName Configuration Domain Name [in]

A\_EE\_CONFIG\_NW =0,

A\_EE\_CONFIG\_FAC =1,

A\_EE\_CONFIG\_UID =2,

A\_EE\_SEED\_KEY = 3,

A\_EE\_CONFIG\_USER = 4,

A\_EE\_CONFIG\_LOCK = 5,

A\_EE\_MEM\_TEST = 6,

A\_EE\_MIDR = 7

* unsigned char \*pPW\_PT Password of Permission[in] 16byte
* unsigned char \*pBuffer Buffer to be read
* int length set to length 64

Function Name: GetconfigArea

Parameter:

* int permType Permission type [in]

G\_PERM\_SUPER\_PASS = 5,

RG\_PERM\_DETOUR\_PASS = 4,

RG\_PERM\_DESTORY0\_PASS = 3,

RG\_PERM\_DESTORY1\_PASS= 2,

RG\_PERM\_EEPROM\_PASS= 1,

RG\_PERM\_UID\_PASS = 0

* int CfgName Configuration Domain name [in]

A\_EE\_CONFIG\_NW =0,

A\_EE\_CONFIG\_FAC =1,

A\_EE\_CONFIG\_UID =2,

A\_EE\_SEED\_KEY = 3,

A\_EE\_CONFIG\_USER = 4,

A\_EE\_CONFIG\_LOCK = 5,

A\_EE\_MEM\_TEST = 6,

A\_EE\_MIDR = 7

* unsigned char \*pPW\_PT Password of Permission[in] 16byte
* unsigned char \*pBuffer Buffer to be read
* int length length of data to be read in bytes