DORCA3 API

내용

[1 SPI MODE 3](#_Toc527365476)

[2 RSA 3](#_Toc527365477)

[2.1 모듈의 정의 3](#_Toc527365478)

[2.2 블록도 4](#_Toc527365479)

[2.3 API 4](#_Toc527365480)

[3 AES/ARIA 5](#_Toc527365481)

[3.1 모듈의 정의 5](#_Toc527365482)

[3.2 블록도 5](#_Toc527365483)

[3.3 API 5](#_Toc527365484)

[3.4 블록 암호화 운영모드 7](#_Toc527365485)

[4 Cipher Memory 13](#_Toc527365486)

[4.1 모듈의 정의 13](#_Toc527365487)

[4.2 블록도 설명 13](#_Toc527365488)

[4.3 API 14](#_Toc527365489)

[5 ECDH 15](#_Toc527365490)

[5.1 모듈의 정의 15](#_Toc527365491)

[5.2 블록도 설명 15](#_Toc527365492)

[5.3 API 16](#_Toc527365493)

[6 PUF&TRNG 17](#_Toc527365494)

[6.1 모듈의 정의 17](#_Toc527365495)

[6.2 블록도 설명 17](#_Toc527365496)

[6.3 API 17](#_Toc527365497)

[7 SHA 18](#_Toc527365498)

[7.1 모듈의 정의 18](#_Toc527365499)

[7.2 블록도 설명 18](#_Toc527365500)

[7.3 API 18](#_Toc527365501)

[8 ECDSA 19](#_Toc527365502)

[8.1 모듈의 정의 19](#_Toc527365503)

[8.2 블록도 설명 19](#_Toc527365504)

[8.3 API 20](#_Toc527365505)

[9 UUP: User Unique Password 21](#_Toc527365506)

[9.1 21](#_Toc527365507)

[9.1 모듈의 정의 21](#_Toc527365508)

[9.2 블록도 설명 21](#_Toc527365509)

[9.3 API 21](#_Toc527365510)

[10 Key 생성 및 변경 22](#_Toc527365511)

[10.1 모듈의 정의 22](#_Toc527365512)

[10.2 블록도 설명 22](#_Toc527365513)

[10.3 API 22](#_Toc527365514)

[11 Setup procedure 23](#_Toc527365515)

[12 Permission 23](#_Toc527365516)

[12.1 모듈의 정의 23](#_Toc527365517)

[12.2 블록도 설명 23](#_Toc527365518)

[12.3 API 23](#_Toc527365519)

[13 Configuration 영역 24](#_Toc527365520)

[13.1 모듈의 정의 24](#_Toc527365521)

[13.2 블록도 설명 24](#_Toc527365522)

[13.3 API 29](#_Toc527365523)

참조

SPI comm. definition

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

1. SPI MODE

비대칭키를 담당하는 CORE 1과 대칭키를 담당하는 CORE 0가 있고 다음과 같이 SPI\_MODE를 설정해야 한다

**CORE 0: 대칭키, Cipher memory, AES etc**

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

**CORE 1: 비대칭키, ECDSA,RSA,ECDH etc**

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

참조

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. 모듈의 정의

RSA 2048 Encryption and Decryption

* 1. 블록도



공개키 알고리즘으로 공개키로 암호화한 데이터를 비밀키로 풀 수 있다

* 1. API

**이름: RsaPubEnc2048**

**기능: RSA Encryption을 수행한다**

**파라메타**

* 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 길이 현재 512로 고정
* int padding [in] 사용하지 않음

**이름: RsaPubDec2048**

**기능: RSA Decryption을 수행한다**

**파라메타**

* 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 길이 현재 512로 고정
* int padding [in] 사용하지 않음

참조

Symmetric-key algorithm

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

1. AES/ARIA
   1. 모듈의 정의

AES128/AES256/ ARIA128/ ARIA256

* 1. 블록도



대칭키 알고리즘으로 암호화와 복호화에 사용되는 키가 같다

* 1. API

**이름: Dorca3CipherDecipher**

**기능: AES 암호화 복호화를 수행한다**

**파라메타**

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

키값이 NULL이면 이전 Key값을 유지한다

* 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 또는 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

**이름: Dorca3CipherDecipherGivenIdx**

**기능: 이전에 저장한 키로 암호화 복호화를 진행한다, 암호화 복호화를 진행 했다면 키는 저장되어 있다.**

**파라메타**

* 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] 사용하지 않음
* 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 또는 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

참조

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. 블록 암호화 운영모드

[암호학](https://ko.wikipedia.org/wiki/%EC%95%94%ED%98%B8%ED%95%99)에서 블록 암호 **운용 방식**(block cipher modes of operation)은 하나의 키 하에서 [블록 암호](https://ko.wikipedia.org/wiki/%EB%B8%94%EB%A1%9D_%EC%95%94%ED%98%B8)를 반복적으로 안전하게 이용하게 하는 절차를 말한다. 블록 암호는 특정한 길이의 블록 단위로 동작하기 때문에, 가변 길이 데이터를 암호화하기 위해서는 먼저 이들을 단위 블록들로 나누어야 하며, 그리고 그 블록들을 어떻게 암호화할지를 정해야 하는데, 이때 블록들의 암호화 방식을 운용 방식으로 부른다.

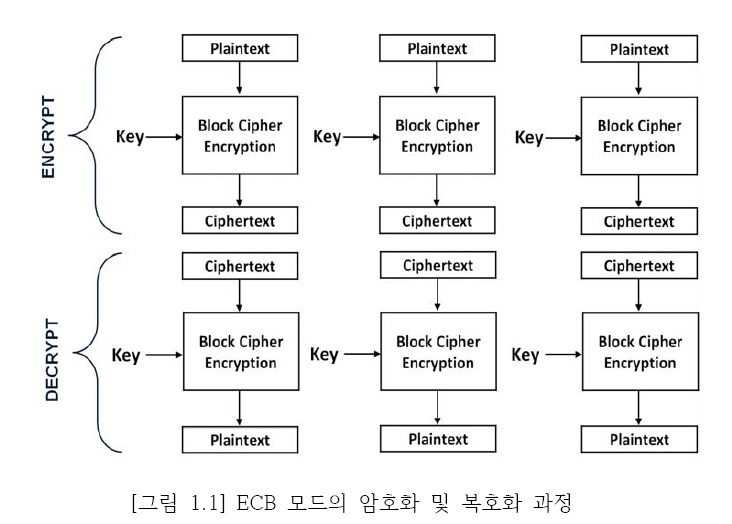
**운영 모드별 기능**

블록암호의 기본 운영모드인 ECB 모드의 보안 취약성을 개선하여 기밀성을 높이기 위한 방법으로 운영모드(mode of operation)가 사용된다. 블록암호의 운영모드는 이전 블록과 초기화벡터(Initialization Vector; IV)의 암호(복호) 결과가 현재 블록의 암호(복호)화에 영향을 미치도록 하여 기밀성을 높이는 방법이다. 대표적인 운영모드로 ECB, CBC,OFB, CFB, CTR, 모드 등이 있으며, 블록간의 의존성과 오류 전파가 평문(복호문)에 미치는 영향이 다르다

1 ECB(Electronic Code Book) 운영모드

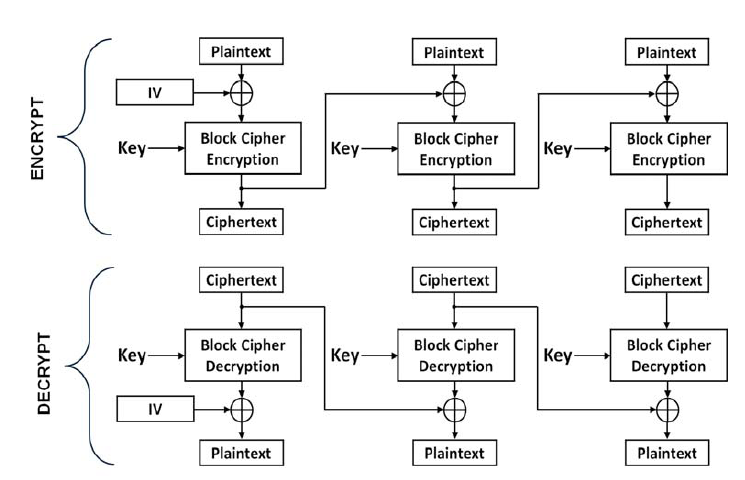
ECB 모드의 암호화(복호화) 과정은 [그림 1.1]과 같으며, 각 블록을 독립적으로 암호 (복호)화 하는 블록암호의 기본 운영모드이다. 메시지의 길이가 평문 길이의 배수가 아닐 경우 패딩을 적용하여 평문 길이의 배수가 되는 블록의 개수를 가지며, 복호화에 사용되는 키는 암호화에 사용되는 키와 동일하다. 키 값이 고정된 경우에, 동일한 평문(암호문)을 암호화(복호화) 하면 동일한 암호문(평문)이 출력되므로, 한 블록이 해독되면 나머지 블록도 해독되는 보안 취약성이 있다. 각 블록이 독립적으로 암호화(복호화)되므로 한 블록에서 오류가 발생해도

다른 블록에 영향을 미치지 않는다.

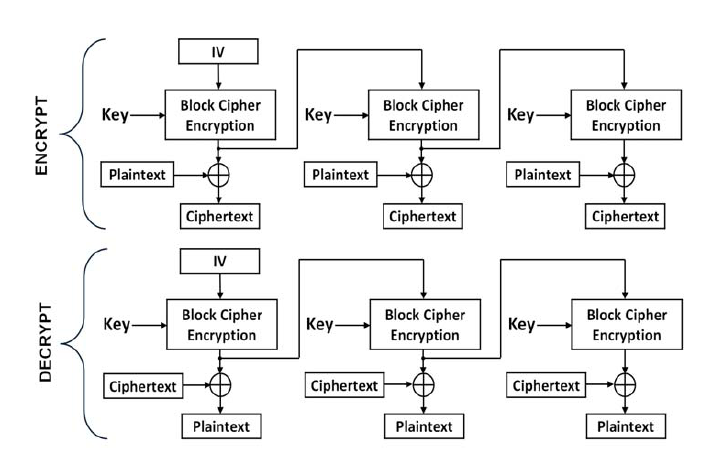


2 CBC(Cipher Block Chaining) 운영모드

CBC 모드의 암호화(복호화) 과정은 [그림 1.2]와 같으며, 각 평문 블록은 이전 블록의 암호화 결과와 XOR 연산된 후, 암호화 된다. 첫번째 블록의 경우에는 초기화 벡터 (IV)가 사용된다. 암호화의 경우 이전 블록의 결과에 의존하므로 병렬처리가 불가능하지만, 복호화의 경우 각 블록을 복호화한 후 이전 블록의 암호문과 XOR 연산하여 복호문이 생성되므로 병렬처리가 가능하다. 이전 블록의 암호문이 현재 블록의 암호화에 영향을 미치므로 동일한 평문에 대해 전혀 다른 암호문이 생성되어 보안성이 높으며, 개인정보를 다루거나 국가기관에서 사용되는 블록암호에 필수적으로 적용된다.

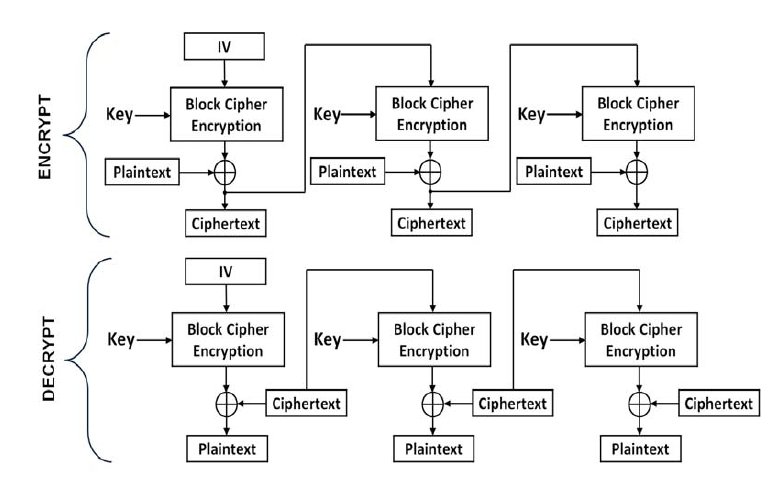


3 OFB(Output FeedBack) 운영모드

OFB 모드의 암호화(복호화) 과정은 [그림 1.3]과 같으며, 이전 블록의 암호화 결과를 현재 블록의 평문(암호문)과 XOR 연산하여 암호문(평문)이 생성된다. 첫 번째 블록의 경우 IV를 암호화한 후, 평문(암호문)과 XOR 연산하여 암호문(평문)으로 출력된다. 암호화와 복호화 과정에서 암호화 연산만 사용되며, 암호화와 복호화의 연산과정 과정이 동일하다.

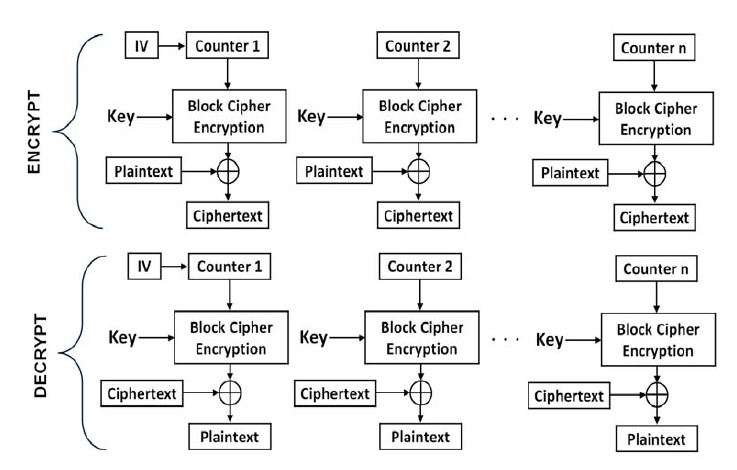
4 CFB(Cipher FeedBack) 운영모드

CFB 모드의 암호화(복호화) 과정은 [그림 1.4]와 같으며, 암호화 과정은 이전 블록의암호문을 암호화한 후 현재 블록의 평문과 XOR 연산하여 암호문으로 출력하고, 복호화과정은 이전 블록의 암호문을 암호화한 후 현재 블록의 암호문과 XOR 연산하여 평문으로 출력한다. 암호화(복호화) 과정 모두 첫 번째 블록의 경우에는 IV를 암호화한 후, 평문(암호문)과 XOR 연산하여 암호문(평문)으로 출력된다. 암호화 및 복호화 연산과정에서 블록암호의 암호연산만 사용되며, CBC 모드와 마찬가지로 암호화는 블록들이 순차적으로 처리되며, 복호화는 병렬처리가 가능하다.



5 CTR(counter) 운영모드

CTR 모드의 암호화(복호화) 과정은 [그림 1.5]와 같으며, 초기화 벡터를 1씩 증가시킨 값을 암호화한 후 평문(암호문)과 XOR 연산하여 암호문(평문)으로 출력된다. 암호화와 복호화 과정이 동일하고, 암호연산만 사용된다. 이전 블록에 대한 의존성을 갖지 않으므로, 다수의 블록에 대한 병렬처리가 가능하여 고속 암호(복호)에 적합하다.



1. Cipher Memory
   1. 모듈의 정의

3840 byte 암호화 메모리

* 1. 블록도 설명



비휘발성 메모리인 eeprom에 데이터를 저장하거나 읽어올 때 사용하는 모듈이며, 사용자가 지정된키를 이용하여 암호화 하여 송신하면 DORCA3은 지정된키로 복호화하여 저장한다

SAMPLE 프로그램에서는 AES\_X0으로 키값을 고정하였다

사용자가 eeprom의 데이터를 읽으면 DORCA3은 지정된 키로 암호화해 결과를 돌려준다

EepromWrite를 이용하여 Cipher text를 write하면 DORCA3은 복호화하여 저장한다

EepromRead를 이용하여 읽으면 DORCA3은 암호화하여 데이터를 리턴한다

eeprom은 PAGE와 SUB PAGE, SUB FRAME으로 나뉘어 지는데 아래의 표와 같다

|  |  |  |
| --- | --- | --- |
|  | 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

**함수명 : EepromWrite**

**설명 :** 지정된키로 암호화된 데이터를 eeprom에 write한다

**파라메터:**

Unsigned char \* Key[in] 암호화에 사용된 키값

키값에 NULL을 입력하면 기존키가 유지된다.

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] 저장할 암호화된 데이터

**함수명 :** **EepromRead**

**설명 :** 입력한 비밀키와 공개키로부터 최종 공통키를 만듬

**파라메터:**

Unsigned char \* Key[in] 복호화에 사용될 키값

키값에 NULL을 입력하면 기존키가 유지된다.

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] 읽어올 암호화된 데이터

참조

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. 모듈의 정의

ECDH P256

* 1. 블록도 설명



두 사용자가 안전하게 비밀이 보장된 공통키를 만드는데 사용한다

위 그림에서 A USER는 자신의 private key DA 를 만들고 DA \* G하여 HA를 만들고 B USER에 전달한다. B USER는 자신의 private key DB 를 만들고 DB \* G하여 HB를 만들고 A USER에 전달한다

최종단계로, A USER는 S = DAHB = DADBG의 공통키를 만든다.

B USER는 S = DBHA = DADBG의 공통키를 만든다.

* 1. API

**함수명 : EcdhGenPubKey**

**설명 :** 입력한 비밀키로부터 공개키를 만듬

**파라메터:**

uint8\_t\* sk[in] 비밀키 length: 32

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

uint8\_t x[32];

uint8\_t y[32];

}

**함수명 : EcdhGenPubKeyPuf**

**설명 :** 랜덤값을 만들고 이를 비밀키로 사용하여 공개키를 만듬

**파라메터:**

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

uint8\_t x[32];

uint8\_t y[32];

}

**함수명 : EcdhGenSessionKey**

**설명 :** 입력한 비밀키와 공개키로부터 최종 공통키를 만듬

**파라메터:**

uint8\_t\* sk[in] 비밀키 length: 32

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

uint8\_t x[32];

uint8\_t y[32];

}

**함수명 : EcdhGenSessionKeyPuf**

**설명 :** 이전에 생성한 비밀키로 사용하여 최종 공통키를 만듬 사용전 ecdh\_gen\_pub\_key\_puf를 미리 호출하여 비밀키를 만들어야함

**파라메터:**

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

uint8\_t x[32];

uint8\_t y[32];

}

**함수명 : KeyFromEcdh**

**설명 :** ECDH를 통해 생성한키를 AES\_X0에 탑재시킨다.

**파라메터:**

없음

참조

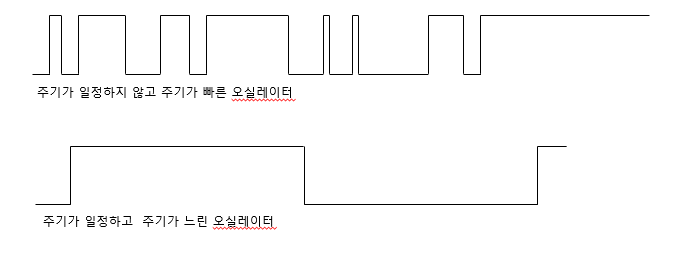
TRNG making:

<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. 모듈의 정의

랜덤한 값을 만든다

* 1. 블록도 설명



주기가 일정하지 않고 주파수가 빠른 주기의 오실레이터를 이를 주기가 일정하고 긴 주기의 오실레이터로 주파수를 카운트하면 카운트 할 때마다 카운트된 숫자가 달라지는 것을 이용하여 난수를 발생시킨다.

* 1. API

**함수명: rand\_pseudorand**

**설명: 랜덤값을 생성한다**

**파라메타**

* char \*out 출력값 16바이트

참조

Hash

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

SHA-2

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

1. SHA
   1. 모듈의 정의

HASH값을 만든다

* 1. 블록도 설명
  2. API

**함수명 : StandardShaMode**

**설명** : HASH값을 만든다

**파라메터**:

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

참조

Elliptic Curve Digital Signature Algorithm

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

1. ECDSA



6. 1. 모듈의 정의

타원곡선 디지털전자 서명 알고리즘

* 1. 블록도 설명



ECDSA는 메시지에 서명을 하여 서명과 메시지를 같이 보내고, 이를 수신한 측에서는 검증을 하여 메시지의 위변조가 없음을 확인하는 알고리즘이다

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

**함수명 : EcdsaGenSignature**

**설명** : 전자서명을 만든다

**파라메터**:

* 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]

**함수명 : EcdsaVerifySignature**

**설명** : 전자서명을 검증한다

**파라메터**:

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]

**함수명 : EcdsaGenPublicKey**

**설명** : 공개키를 만든다

**파라메터**:

unsigned char \*private\_key length:32 비밀키[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. 모듈의 정의

해당 칩의 유일한 구별 번호인 User Unique Password에 관련한 내용이다. 본래 이 모듈의 명칭은 Root Serial이였으나, 잘못된 해석으로 인해 백도어의 우려를 간구해서 명칭을 UUP: User Unique Password로 수정하였다. 실제로 이 모듈은 사용자가 정하는 고유 키/비밀번호다.

* 1. 블록도 설명

4개의 Serial을 저장하며, Serial을 읽을 때는 SHA한 결과를 읽는다

UID 권한을 사용하면 1번,3번 Serial의 원본을 읽을 수 있다

* 1. API

**함수명** : **RSCreate01**

**설명** : 네오와인 알고리즘으로 디바이스별로 고유한 serial 0,1을 생성한다

**파라메터:** 없음

**함수명** : **RSCreate23**

**설명** : 사용자가 입력한 값으로 serial 2,3을 생성한다

**파라메터:**

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

**함수명 : RSSHAReadIdx**

**설명** : serial을 SHA해서 읽어온다

**파라메터:**

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

**함수명: RSDirectRead13**

**설명:** UID PASS WORD를 입력하여 1번,3번 Serial의 원본을 읽는다

**파라페터:**

unsigned char \*pRS: output [out]

int index: index값 1또는 3 [in]

unsigned char \*UID\_PW\_PT: 16바이트 UID 비밀번호 포인터[in]

1. Key 생성 및 변경
   1. 모듈의 정의

암복호화에 사용될 Key값을 생성하거나 변경하는 방법이다

* 1. 블록도 설명



키는 SEED\_KEY, AES\_KEY\_0,AES\_KEY\_1, AES\_KEY\_2, AES\_KEY\_3로 계층화 되어있고 SEED\_KEY는 출시할 때 write하며, SEED\_KEY를 이용하여 AES\_KEY0을 만들고 AES\_KEY1를 이용하여 AES\_KEY2를 만든다 AES\_KEY2를 이용하여 AES\_KEY3를 만든다.

샘플 프로그램에서는 AES\_KEY\_1을 permission get 동작에 사용하고 AES\_KEY0를 암복호화에 사용한다.

* 1. API

**함수명: KeyCreatXn**

**설명**: Key를 생성하거나 변경한다

**파라메터:**

* int index: Key index[in]
* int mode: MODE256 또는 MODE128[in]
* unsigned char \*prevKey: Key 계층구조의 이전의 Key값[in]
* unsigned char \*key: 새로 생성하거나 변경하는 Key값[in]

1. Setup procedure



SEED KEY는 출하단계에서 Writing이 된다. 그 이후 SEED KEY를 이용하여 AES KEY를 만든다.

ROOT SERIAL은 AES KEY를 생성한 이후에 만든다

download 파일을 실행하면 writing이 된다

1. Permission
   1. 모듈의 정의

Permission을 가지고 있으면 특정 기능을 수행하거나, 데이터를 읽을 수 있다

Permission은 SUPER, DETOUR, DESTORY0, DESTORY1, EEPROM, UID 가 있다

* 1. 블록도 설명

생략

* 1. API

**함수명: GetPermission**

**설명 :** Permission을 획득한다

**파라메터**

* 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 권한에 해당하는 비밀번호[in]
* int index 퍼미션을 얻을 때 사용하는 key index[in]

**함수명 ReleasePermision**

**설명:** Permission을 해제한다

**퍼라메터:** 없음

1. Configuration 영역
   1. 모듈의 정의

칩을 설정하기 위한 영역이다 상세한 설명은 추후에 보완하겠다

* 1. 블록도 설명

Configuration memory는 아래 종류별로 구분 할 수 있다.. 각 영역 별로 아래와 같은 특성을 가지고 있다.

* EE\_CONFIG\_UID

EE\_UID\_PW 퍼미션을 획득하면 API로 수정할 수 있다

* EE\_SEED\_KEY

Key Memory 설명 참고

EE\_UID\_PW 퍼미션을 획득하면 API로 수정할 수 있다

* EE\_CONFIG\_LOCK

EE EE\_UID\_PW 퍼미션을 획득하면 API로 수정할 수 있다

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

EE\_SEED\_KEY\_LOCK

EE\_KEY\_AES\_LOCK

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | EE\_CONFIG\_UID  0xEBC0 | | | | | |  |  | EE\_CONFIG\_UID 사용자가 설정하는 값이다. EE\_CONFIG\_LOCK:EE\_CONFIG\_UID\_LOCK에 의해 lock되면 수정 할 수 없다. |
| 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. 사용자가 사용법을 설계하여 사용한다. EE\_UID\_PW로 접근 제한 되어야 한다. 0x4444\_4444 |
| EE\_CONFIG\_USER\_CTRL1 | 1 | [7:2] Reserved [1] EE\_CLKXTAL\_EN 0 : CLKXTAL이 high, low 토글하는 정상 상태이다. (default) 1 : CLKXTAL이 high 상태를 유지한다.  [0] EE\_OSC\_5M\_20M 0 : 20 MHz (default) 1 : 5 MHz |

|  |  |  |  |
| --- | --- | --- | --- |
| EE\_SEED  0xEC00 | EE\_SEED\_KEY | 32 | 사용자(E-MCU)가 UID\_PW permission을 획득하여 관리자 모드에서(EE\_CONFIG\_LOCK:EE\_SEED\_KEY\_LOCK가 unlock 상태) 이 영역을 write/read 할 수 있다. EE\_CONFIG\_LOCK:EE\_SEED\_KEY\_LOCK가 Lock 되면 사용자(E-MCU)는 UID\_PW permission을 획득하여도, EE\_SEED\_KEY를 write/read 할 수 없다. 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 | E-MCU는 ST0\_EE\_CFG state에서 이 페이지에 write 할 수 없다. E-MCU는 ST0\_EE\_CFG state에서 이 페이지를 자유롭게 읽어 볼 수 있다. [7:3] Reserved [2] EE\_RS\_x2\_FLAG  0 : EE\_RS\_x2이 만들어지기 전 상태이다. 1 : EE\_RX\_x2이 만들어진 상태이다. [1:0] :RESERVED |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | EE\_CONFIG\_LOCK  0xEC80 | | | | | EE\_UZx0\_WR\_LOCK ~  EE\_UZxF\_WR\_LOCK | 16 | EE\_UID\_PW permission 얻어야 이영역 bit set(0->1) 가능하다. OTP(One Time Programmable) 영역이다. 사용자(E-MCU)가 각 비트의 0 값을 1 로 변경할 수는 이지만, 값을 1에서 0 으로 변경 불가능하다. EE\_USER\_ZONE\_M00 ~ EE\_USER\_ZONE\_M15 16개 중 1개 영역에 매칭 된다.  EE\_UZx0\_WR\_LOCK -> EE\_USER\_ZONE\_M11에 매칭된다. Lock 되어 있으면 write 금지된다. 0x00 : EE\_UZxN영역이 Unlock 상태이다. 0x01~0xFF : EE\_UZxN영역이 Lock 된다. 변경 불가 하다. |
| EE\_SEED\_KEY\_LOCK | 1 | EE\_UID\_PW permission 얻어야 이영역 bit set(0->1) 가능하다. OTP(One Time Programmable) 영역이다. 사용자(E-MCU)가 각 비트의 0 값을 1 로 변경할 수는 이지만, 값을 1에서 0 으로 변경 불가능하다. 0x00 : EE\_SEED\_KEY영역이 Unlock 상태이다. 0x01~0xFF : EE\_SEEK\_KEY영역이 Lock 된다. 변경 불가 하다. |
| EE\_RS\_x0 LOCK | 1 | EE\_UID\_PW permission 얻어야 이영역 bit set(0->1) 가능하다. OTP(One Time Programmable) 영역이다. 사용자(E-MCU)가 각 비트의 0 값을 1 로 변경할 수는 이지만, 값을 1에서 0 으로 변경 불가능하다. 0x00 : EE\_RS\_x0영역이 Unlock 상태이다. 0x01~0xFF : EE\_RS\_x0영역이 Lock 된다. 변경 불가 하다. |
| EE\_KEY\_AES\_LOCK | 1 | EE\_UID\_PW permission 얻어야 이영역 bit set(0->1) 가능하다. OTP(One Time Programmable) 영역이다. 사용자(E-MCU)가 각 비트의 0 값을 1 로 변경할 수는 이지만, 값을 1에서 0 으로 변경 불가능하다. 0x00 : EE\_KEY\_AES\_xN 영역이 Unlock 상태이다. E-MCU가 EE\_KEY\_AES\_xN 영역을 읽어 볼 수 있다. 0x01~0xFF : EE\_KEY\_AES\_xN 영역이 Lock 된다. E-MCU가 EE\_AES\_xN 영역을 읽어 볼 수 없다. |

* 1. API

함수명: SetConifgArea

파라메터:

* int permType Permission type [in]

RG\_PERM\_EEPROM\_PASS= 1,

RG\_PERM\_UID\_PASS = 0

* int CfgName Configuration 영역 이름 [in]

A\_EE\_CONFIG\_UID =2,

A\_EE\_SEED\_KEY = 3,

A\_EE\_CONFIG\_USER = 4,

A\_EE\_CONFIG\_LOCK = 5,

unsigned char \*pPW\_PT Permission에 해당하는 비밀번호[in] 16byte

* unsigned char \*pBuffer Write할 버퍼 64바이트
* int length 64로 고정

함수명: GetConfigArea

파라메터:

* int permType Permission type [in]

RG\_PERM\_EEPROM\_PASS= 1,

RG\_PERM\_UID\_PASS = 0

* int CfgName Configuration 영역 이름 [in]

A\_EE\_CONFIG\_UID =2,

A\_EE\_SEED\_KEY = 3,

A\_EE\_CONFIG\_USER = 4,

A\_EE\_CONFIG\_LOCK = 5,

unsigned char \*pPW\_PT Permission에 해당하는 비밀번호[in] 16byte

* unsigned char \*pBuffer read할 버퍼
* int length 읽어올 데이터의 길이 in byte