SEED 소스코드 매뉴얼 v2.0

2009. 8.

- 차 례 -

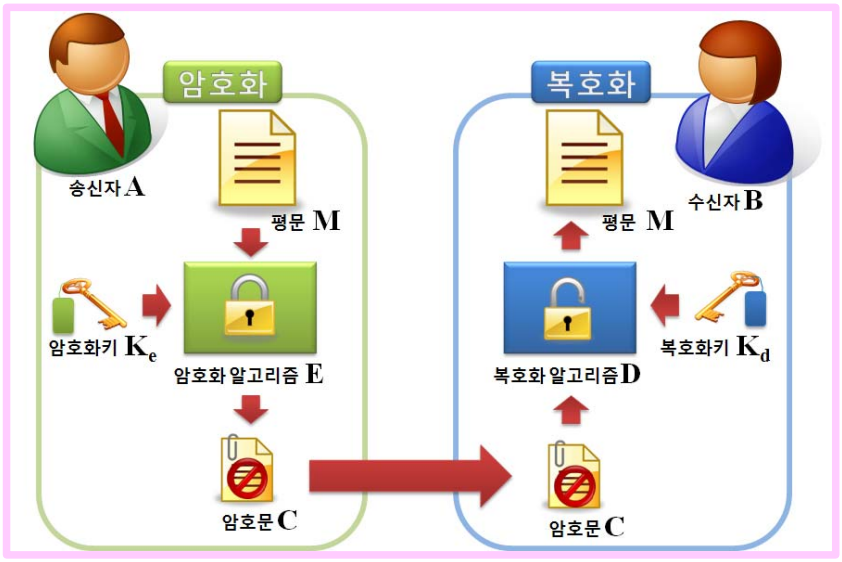
1. 개요

SEED(SEED 128 혹은 SEED 256 등으로 지칭하지 않는 이상, SEED 128/256 모두를 의미)는 전자상거래, 금융, 무선통신 등에서 전송되는 중요 정보를 보호하기 위해 2009년 한국인터넷진흥원(KISA, (구) 한국정보보호진흥원)을 중심으로 국내 암호 전문가들이 참여하여 순수 국내기술로 개발한 블록암호 알고리즘이다.

본 매뉴얼은 SEED 소스코드를 보다 쉽게 활용할 수 있도록 SEED 소스코드에 대한 설명과 함께 소스코드 사용 시 주의사항을 다룬 매뉴얼이다.

2. 블록암호 알고리즘 SEED

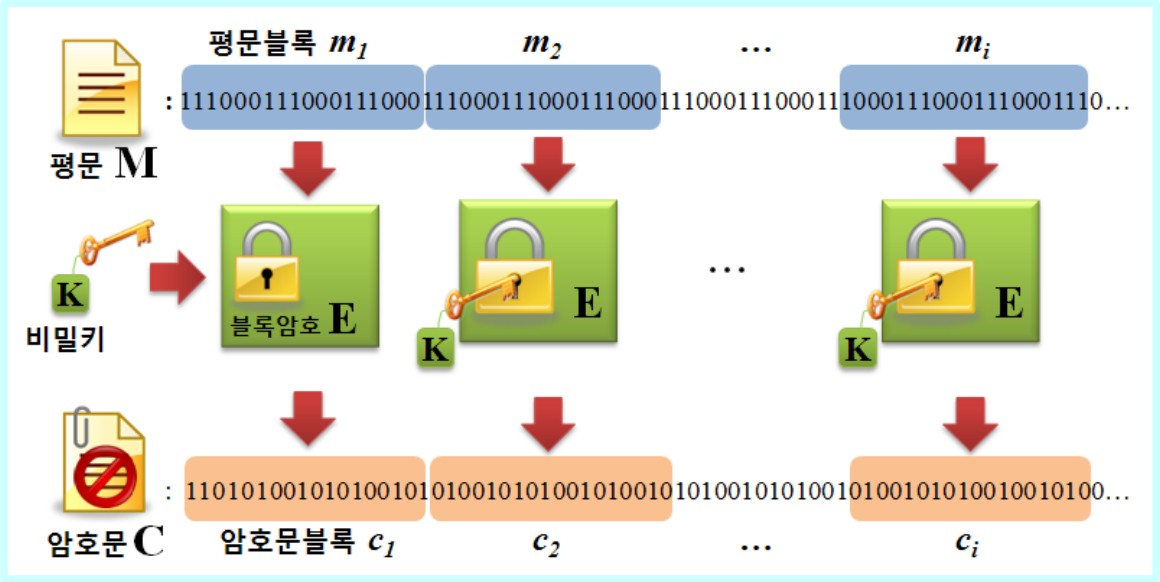
암호(Cryptography)란 메시지를 해독 불가능한 형태로 변환하거나 또는 암호화된 메시지를 해독 가능한 형태로 변환하는 기술을 말한다. 이때, 해독 가능한 형태의 메시지를 **평문**(Plaintext)이라고 하고, 해독 불가능한 형태의 메시지를 **암호문**(Ciphertext)이라 하며, 평문을 암호문으로 변환하는 과정을 **암호화**(Encryption), 암호문을 평문으로 변환하는 과정을 **복호화**(Decryption)라고 한다. 정보를 숨기고, 숨긴 정보를 볼 수 있기 위해서는 권한이 있는 사람만이 암호화 및 복호화를 할 수 있어야 한다. 그래서 두 사람은 제 삼자가 모르는 비밀 정보를 공유해야 하고 이 정보는 암호화 및 복호화에서 매우 중요한 역할을 담당한다.



[그림 2-1] 암・복호화 과정

이러한 비밀 정보를 우리는 **비밀키**(Secret Key)라고 하며, 암호화에 필요한 **암호화키**(Encryption Key)와 복호화에 필요한 **복호화키**(Decryption Key)로 분류한다. 일반적으로 암호화 및 복호화 과정은 [그림 2-1]과 같다.

암호는 키의 특성에 따라, 암호화키와 복호화키가 같은 암호를 **대칭키 암호**(또는 **비밀키 암호**)라고 하며, 암호화키와 복호화키가 다른 암호를 **비대칭키 암호**(또는 공개키 암호)라고 한다. 또한, 대칭키 암호는 다시 암·복호화를 처리하는 방식에 따라 메시지를 블록단위로 나누어 처리하는 **블록암호**와 메시지를 비트단위로 처리하는 **스트림암호**로 분류한다.



[그림 2-2] 블록암호 암호화(ECB모드) 과정

SEED 128은 128비트, SEED 256은 256비트의 암·복호화키를 이용하여 임의의 길이를 갖는 입력 메시지를 128비트의 블록단위로 암·복호화를 처리하는 블록암호 알고리즘이다. 따라서 임의의 길이를 가지는 평문 메시지를 128비트씩 블록단위로 나누어 암호화하여 128 비트 암호문을 생성한다.

3. SEED 구현 시 고려사항

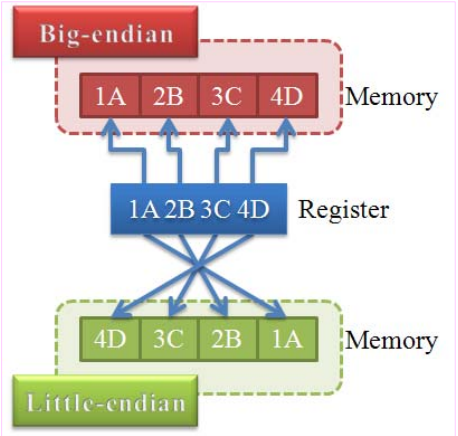
블록암호 SEED를 실제로 사용하기 위해서는 구현 환경에서 사용되는 **엔디안 (endianness)**과 암호화된 데이터가 저장되는 **데이터형식(Data Type)**을 고려해야 한다. 또한, 입력 블록을 블록암호에 적용하는 방법인 **운영모드(Mode of operation)**와 마지막 블록의 크기를 맞추기 위한 **패딩(Paddingg)**을 함께 구현해 주어야 한다.

3.1. 엔디안

엔디안이란, 컴퓨터 메모리의 바이트를 배열하는 순서를 말한다. 엔디안은 보통 큰 단위가 앞에 나오는 빅 엔디안(Big-endian)과 작은 단위가 앞에 나오는 리틀 엔디안(Little-endian), 그리고 두 경우에 속하지 않거나 둘 모두를 지원하는 미들 엔디안(Middle-endian)으로 분류한다. 현재 빅 엔디안과 리틀 엔디안이 많이 사용되나, x86 아키텍처가 리틀 엔디안을 쓰기 때문에 대부분의 x86 아키텍처를 사용하는 데스크탑은 리틀 엔디안을 사용하며, **SEED 소스코드도 기본적으로 리틀 엔디안을 사용**하고 있다.

종류	0x1A2B의 표현	0x1A2B3C4D의 표현
빅 엔디안	1A 2B	1A 2B 3C 4D
리틀 엔디안	2B 1A	4D 3C 2B 1A
미들 엔디안	-	2B 1A 4D 3C 또는 3C 4D 1A 2B

<표 2-1> 엔디안에 따른 표현



엔디안은 암호 알고리즘 구현 시에 중요하게 고려되어야 할 문제이다. 대부분의 암호 알고리즘이 비트 단위 연산을 처리하기 때문에, 바이트 배열이 서로 맞지 않으면 같은 알고리즘으로 같은 메시지를 암호화하더라도 서로 다른 암호문을 생성할 수 있다.

```
#if __alpha__ || __alpha | __i386__ || i386 || _M_I86 || _M_IX86 || ₩
    __OS2__ || sun386 || __TURBOC__ || vax || vms || VMS || __VMS
#define SEED_LITTLE_ENDIAN
#else
#define SEED_BIG_ENDIAN
#endif
```

현재 배포 중인 **C 소스코드**는 자동으로 시스템이 사용하는 엔디안을 판별하도록 되어 있다. 그러나 사용자의 환경에 맞지 않는 경우에는 수동으로 올바른 엔디안을 설정하여 사용한다.

자바 가상 머신은 기본적으로 빅 엔디안을 사용하기 때문에, **Java 소스코드**에서는 기본 엔디안으로 빅엔디안이 설정되어 있으나, 상황에 따라 사용자는 시스템의 엔디안에 맞게 수동으로 선택해 주어야 한다.

```
private static Boolean LITTLE      = false;
private static Boolean BIG         = true;

private static Boolean ENDIAN      = BIG;     // Java virtual machine uses big endian as a default
//private static Boolean ENDIAN    = LITTLE;
```

3.2. 데이터 형식

일반적으로 암호 알고리즘은 비트단위 연산을 포함한 다양한 연산을 수행하여 평문 메시지를 무의미한 비트열인 암호문으로 변환한다. 이때 만들어진 암호문 비트열은 랜덤한 비트들로 구성되기 때문에 이를 문자열(string)의 형태로 저장하거나 처리할 경우에는 문제가 발생할 수 있다.

예를 들어, 다음의 C 소스코드를 보면, ch배열에 저장되는 값은 0x41, 0x42, 0x00, 0x44, 0x45 이지만, 이를 스트링으로 출력할 경우, ch[3] = 0x00 = NULL 값에 의해 0x41, 0x42에 해당되는 "AB"만 출력이 된다.

```
char ch[] = {0x41, 0x42, 0x00, 0x44, 0x45};
printf("%s", ch);
```

따라서 **암호화된 메시지는 항상 문자열이 아닌 이진(binary) 형태로 처리**해 주어야 한다.

3.3. 운영모드

운영모드란, 여러 개의 입력 블록들을 블록암호에 적용하여 암·복호화하는 방법에 대한 정의이다. 이러한 운영모드는 블록암호와는 독립적으로 정의된다. 대표적으로 가장 널리 이용되는 블록암호 운영모드에는 ECB(Electronic Code Book) 모드, CBC(Cipher Block Chaining) 모드, CFB(Ciphertext FeedBack) 모드, OFB(Output FeedBack) 모드, CTR(CounTeR) 모드가 있다. SEED에 대한 운영모드는 SEED 홈페이지에서 제공하는 "**다양한 보안 프로토콜에서의 SEED 이용 가이드라인**"을 참고할 수 있다.

3.4. 패딩

메시지를 SEED에 입력하기 위해 여러 개의 128비트 블록으로 나눌 때, 마지막 블록이 정확히 그 크기를 맞추는 것은 쉽지 않을 것이다. 예를 들어, 300비트의 메시지를 128비트의 블록으로 나눌 경우, 300 = 128 + 128 + 44 로 나뉘어 세 개의 블록을 구성하게 되는 데, 이때 마지막 블록이 44비트로 128비트를 만족하지 못한다. 이 경우 나머지 부족한 84비트를 채워주어야 SEED의 입력 값으로 사용할 수 있다. 이렇게 부족한 부분을 채우는 방식을 패딩이라고 한다. SEED에 주로 사용되는 패딩은 SEED 홈페이지에서 제공하는 "**다양한 보안프로토콜에서의 SEED 이용 가이드라인**"을 참고할 수 있다.

4. C 소스코드

4.1. 구성요소

배포되는 소스코드는 다음과 같이 5개의 파일로 구성되어 있다.

4.1.1. SEED 128

- **SEED_KISA.c**
  SEED의 주요 함수 SeedEncrypt, SeedDecrypt, SeedRoundKey를 정의
- **SEED_KISA.h**
  SEED에 사용되는 매크로, 상수, 데이터형 등을 정의한 헤더파일
- **SEED_KISA.tab**
  SEED에 사용되는 S-Box 테이블을 정의
- **SEED_test_KISA.c**
  SEED가 제대로 동작하는 지를 테스트하기 위한 파일
- **test vector.txt**
  SEED를 사용하여 얻은 테스트 벡터로, 제대로 동작하는 지를 확인할 때 활용

4.1.2. SEED 256

- **SEED_256_KISA.c**
  SEED의 주요 함수 SeedEncrypt, SeedDecrypt, SeedRoundKey를 정의
- **SEED_256_KISA.h**
  SEED에 사용되는 매크로, 상수, 데이터형 등을 정의한 헤더파일
- **SEED_256_KISA.tab**
  SEED에 사용되는 S-Box 테이블을 정의
- **SEED_256_test_KISA.c**
  SEED가 제대로 동작하는 지를 테스트하기 위한 파일
- **SEED_256_testvector.hwp**
  SEED를 사용하여 얻은 테스트 벡터로, 제대로 동작하는 지를 확인할 때 활용

4.2. 함수 설명

SEED를 이용하여 암호화 및 복호화를 수행하기 위해서 다음의 함수들을 사용한다.

■ SeedEncrypt

SeedEncrypt 함수는 SEED를 이용하여 암호화를 수행한다.

```
void SeedEncrypt (
        BYTE    *pbData,
        DWORD *pdwRoundKey
);
```

매개변수

**pbData*

[in, out] 128비트의 암호화될 데이터를 입력받아 128비트의 암호화된 데이터를 출력

**pdwRoundKey*

[in] SEED의 각 라운드에 사용되는 라운드 키를 입력. 라운드 키는 SeedRoundKey 함수를 통해 사용자의 비밀키로부터 생성

■ SeedDecrypt

SeedDecrypt 함수는 SEED를 이용하여 복호화를 수행한다.

```
void SeedDecrypt (
        BYTE    *pbData,
        DWORD *pdwRoundKey
);
```

매개변수

**pbData*

[in, out] 128비트의 암호화된 데이터를 입력받아 128비트의 복호화된 데이터를 출력

**pdwRoundKey*

[in] SEED의 각 라운드에 사용되는 라운드 키를 입력. 라운드 키는 SeedRoundKey 함수를 통해 사용자의 비밀키로부터 생성

■ SeedRoundKey

SeedRoundKey 함수는 사용자의 비밀키를 이용하여 SEED의 암호화 및 복호화에 사용되는 라운드 키를 생성한다.

```
void SeedRoundKey(
        DWORD *pdwRoundKey,
        BYTE  *pbUserKey
);
```

매개변수

**pdwRoundKey*

[out] SEED의 각 라운드에 사용되는 라운드 키

**pbUserKey*

[in] 라운드 키를 생성하기 위해 사용되는 사용자의 비밀키

4.3. SEED 사용 예제

4.3.1. SEED 128

```
DWORD pdwRoundKey[32];
BYTE    pbUserKey[16]  =  {0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,
                           0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00};
BYTE    pbData[16]     =  {0x00,  0x01,  0x02,  0x03,  0x04,  0x05,  0x06,  0x07,
                           0x08,  0x09,  0x0A,  0x0B,  0x0C,  0x0D,  0x0E,  0x0F};

/* Derive roundkeys from user secret key */
SeedRoundKey(
        pdwRoundKey,
        pbUserKey
);

/* Encryption */
SeedEncrypt(
        pbData,
        pdwRoundKey
);

/* Decryption */
SeedDecrypt(
        pbData,
        pdwRoundKey
);
```

4.3.2. SEED 256

```
DWORD pdwRoundKey[48];
BYTE    pbUserKey[32] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
BYTE    pbData[16]    = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                         0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};

/* Derive roundkeys from user secret key */
SeedRoundKey(
        pdwRoundKey,
        pbUserKey
);

/* Encryption */
SeedEncrypt(
        pbData,
        pdwRoundKey
);

/* Decryption */
SeedDecrypt(
        pbData,
        pdwRoundKey
);
```