

# SEED 블록암호 알고리즘에 대한 소스코드 활용 매뉴얼

2013.12.

**KISA** 한국인터넷진흥원  
Korea Internet & Security Agency

## 제 · 개 정 이 력

순번	제 · 개정일	제 · 개정 내역	담당자
1	2013.12	“SEED 블록암호 알고리즘에 대한 소스코드 활용” 매뉴얼	김기문
2			
3			
4			

## < 목 차 >

1. 개요 .....	5
2. 블록암호 알고리즘 .....	5
3. 고려 사항 .....	6
3.1 엔디안 .....	6
3.2 데이터 형식 .....	7
3.3 운영 모드 .....	8
3.3.1 ECB 운영모드 .....	8
3.3.2 CBC 운영모드 .....	9
3.3.3 CTR 운영모드 .....	10
3.4 패딩 방법 .....	10
3.4.1 패딩 방법 1 .....	11
3.4.2 패딩 방법 2 .....	11
3.4.3 패딩 방법 3 .....	12
3.5 활용 방법 .....	12
4. 응용 프로그램 .....	13
4.1 C/C++ .....	13
4.1.1 프로젝트 생성 및 빌드 .....	13
4.1.2 소스코드 설명 .....	17
4.2 Java .....	25
4.2.1 프로젝트 생성 및 빌드 .....	25
4.2.2 소스코드 설명 .....	28

5. 웹 프로그램 .....	36
5.1 ASP .....	36
5.1.1 소스코드 추가 .....	36
5.1.2 소스코드 설명 .....	37
5.2 JSP .....	45
5.2.1 소스코드 추가 .....	45
5.2.2 소스코드 설명 .....	46
5.3 PHP .....	54
5.3.1 소스코드 추가 .....	54
5.3.2 소스코드 설명 .....	55
<b>[부록] 참조구현값 .....</b>	<b>63</b>
1. ECB 모드 참조구현값 .....	64
2. CBC 모드 참조구현값 .....	66
3. CTR 모드 참조구현값 .....	68

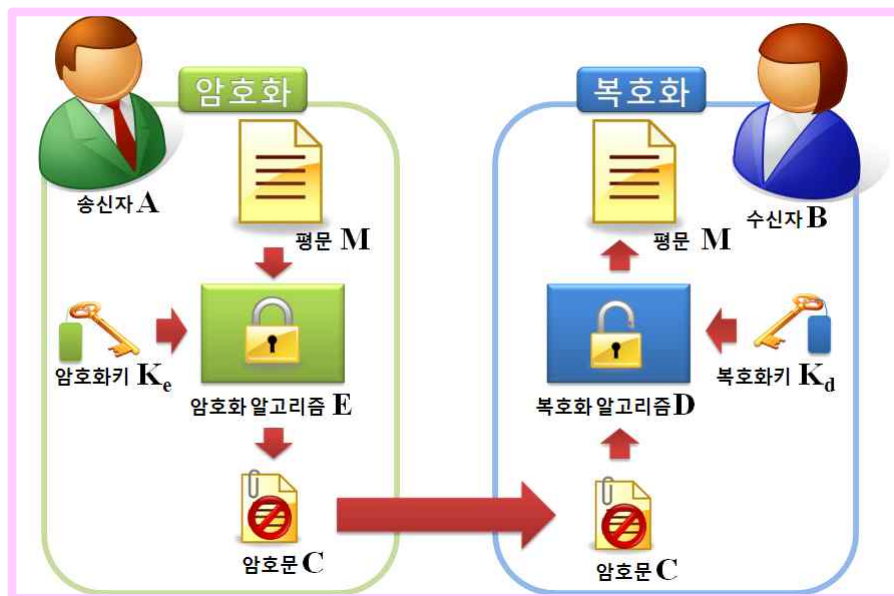
## 1. 개요

SEED는 전자상거래, 금융, 무선통신 등에서 전송되는 중요 정보를 보호하기 위해 1999년 2월 한국 정보보호진흥원을 중심으로 국내 암호 전문가들이 참여하여 순수 국내기술로 개발한 블록암호 알고리즘이다.

본 매뉴얼은 SEED 운영모드별 소스코드를 다양한 언어 및 플랫폼에서 쉽게 활용할 수 있도록 C/C++, Java, ASP, PHP, JSP용으로 개발된 소스코드의 설명과 함께 사용 시 주의사항을 다룬다.

## 2. 블록암호 알고리즘

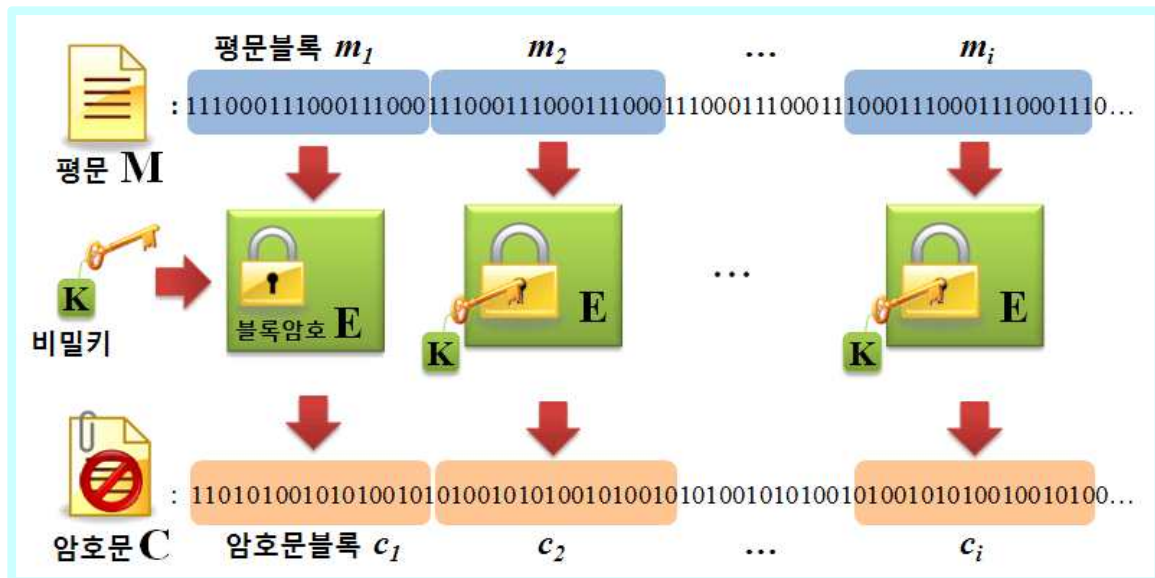
암호(Cryptography)란 메시지를 해독 불가능한 형태로 변환하거나 또는 암호화된 메시지를 해독 가능한 형태로 변환하는 기술을 말한다. 이때, 해독 가능한 형태의 메시지를 평문(Plaintext)이라고 하고, 해독 불가능한 형태의 메시지를 암호문(Ciphertext)이라 하며, 평문을 암호문으로 변환하는 과정을 암호화(Encryption), 암호문을 평문으로 변환하는 과정을 복호화(Decryption)라고 한다. 정보를 숨기고, 숨긴 정보를 볼 수 있기 위해서는 권한이 있는 사람만이 암호화 및 복호화를 할 수 있어야 한다. 그래서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



< 암호화·복호화 과정 >

암호는 키의 특성에 따라, 암호화키와 복호화키가 같은 암호를 대칭키 암호(또는 비밀키 암호)라고 하

며, 암호화키와 복호화키가 다른 암호를 비대칭키 암호(또는 공개키 암호)라고 한다. 또한, 대칭키 암호는 다시 암·복호화를 처리하는 방식에 따라 메시지를 블록단위로 나누어 처리하는 블록암호와 메시지를 비트단위로 처리하는 스트림암호로 분류한다.



< 블록암호 암호화(ECB모드) 과정 >

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

### 3. 고려사항

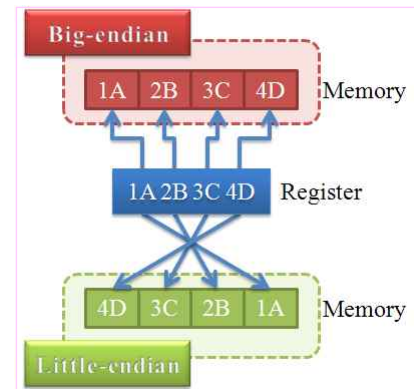
블록암호 SEED를 실제로 사용하기 위해서는 구현 환경에서 사용되는 엔디안(endianness)과 암호화된 데이터가 저장되는 데이터 형식(Data Type)을 고려해야 한다. 또한, 입력 블록을 블록암호에 적용하는 방법인 운영모드(Mode of operation)와 마지막 블록의 크기를 맞추기 위한 패딩(Padding)을 함께 구현해 주어야 한다.

#### 3.1. 엔디안

엔디안이란, 컴퓨터 메모리에 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 많은 시스템에서 빅 엔디안과 리틀 엔디안이 많이 사용되고 있다. 일반적인 데스크톱에서는 x86 아키텍처를 많이 사용하며, x86은 리틀 엔디안의 구조를 사용하기 때문에, SEED 소스코드도 기본적으로 리틀 엔디안을 사용하여 설명한다.

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

< 엔디언에 따른 표현 >



엔디언은 암호 알고리즘 구현 시에 중요하게 고려되어야 할 문제인데, 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 || W
    __OS2__ || sun386 || __TURBOC__ || vax || vms || VMS || __VMS
#define LITTLE_ENDIAN
#else
#define BIG_ENDIAN
#endif
```

자바 가상 머신은 기본적으로 빅 엔디언을 사용하기 때문에, Java 소스코드에서는 기본 엔디언으로 빅 엔디언이 설정되어 있으나, 상황에 따라 사용자는 시스템의 엔디언에 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 HEX 형태로 처리해 주어야 한다.

### 3.3. 운영모드

운영모드란, 여러 개의 입력 블록들을 블록암호에 적용하여 암호·복호화하는 방법에 대한 정의이다. 이러한 운영모드는 블록암호와 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. 하지만 이번 매뉴얼에서는 KISA에서 개발하여 배포하는 ECB, CBC, CTR에 대해서만 설명을 하도록 하겠다.

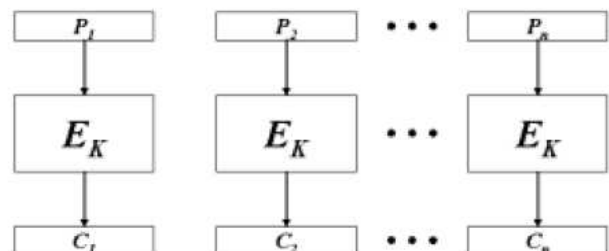
#### 3.3.1 ECB 운영모드

ECB(Electronic Codebook) 모드는 평문 블록을 암호문 블록으로 독립적으로 암호화하는 운영 모드이다.

ECB 모드의 암호화 과정은 평문 블록( $P_i$ )을 입력 블록( $I_i$ )으로 설정하고, 이를 암호화한 출력 블록( $O_i$ )을 암호문 블록( $C_i$ )으로 설정한다.

```
o 입력 :  $P=\{P_1, \dots, P_n\}$ ,  $K$ 
o 출력 :  $C=\{C_1, \dots, C_n\}$ 
o 처리과정
  for  $i=1$  to  $n$ 
     $I_i = P_i$ 
     $O_i = E_K(I_i)$ 
     $C_i = O_i$ 
```

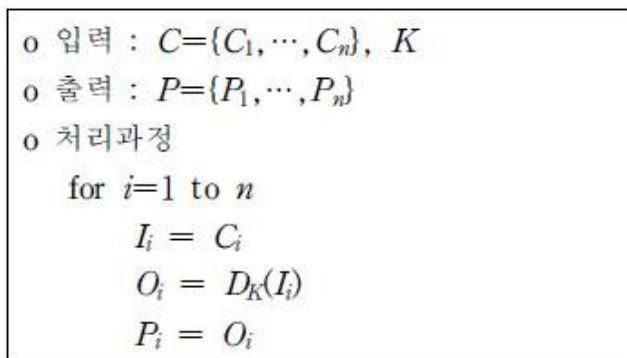
< ECB 암호화 의사코드 >



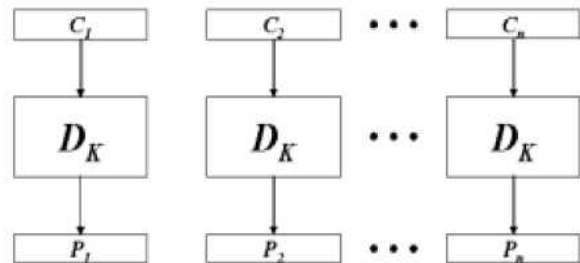
< ECB 암호화 >

ECB 모드의 복호화 과정은 평문 블록( $C_i$ )을 입력 블록( $I_i$ )으로 설정하고, 이를 암호화한 출력 블록( $O_i$ )을 암호문 블록( $P_i$ )으로 설정한다.





< ECB 복호화 의사코드 >

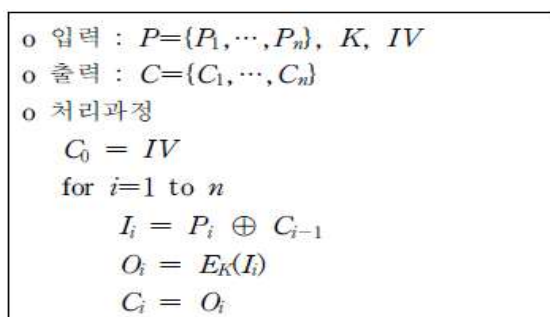


< ECB 복호화 >

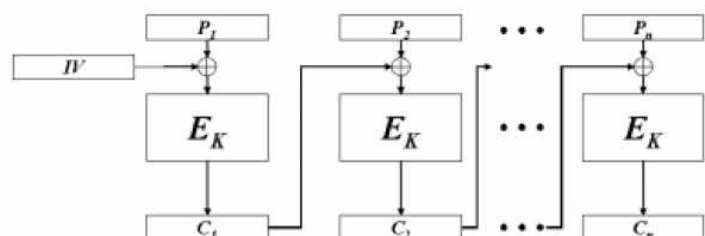
### 3.3.2 CBC 운영모드

CBC(Cipher Block Chaining) 모드는 동일한 평문 블록과 암호문 블록 쌍이 발생하지 않도록 전 단계의 암호화 결과가 현 단계에 영향을 주는 운영모드이다

CBC 모드의 암호화 과정은 현 단계에서 평문 블록( $P_i$ )과 전 단계의 암호문 블록( $C_{i-1}$ )을 배타적 논리합 연산한 결과를 현 단계의 입력 블록( $I_i$ )으로 설정하고, 이를 암호화한 출력 블록( $O_i$ )을 현 단계의 암호문 블록( $C_i$ )으로 설정한다.



< CBC 암호화 의사코드 >

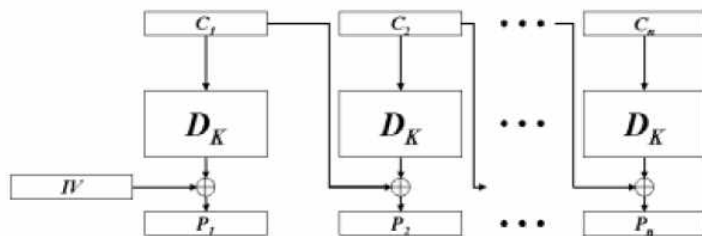


< CBC 암호화 >

CBC 모드의 복호화 과정은 현 단계의 암호문 블록( $C_i$ )을 입력 블록( $I_i$ )으로 설정하고, 이를 복호화한 출력 블록( $O_i$ )을 전 단계의 입력 블록( $I_{i-1}$ )인 암호문 블록( $C_{i-1}$ )과 배타적 논리합 연산한 결과를 현 단계의 평문 블록( $P_i$ )으로 한다.

o 입력 :  $C=\{C_1, \dots, C_n\}$ ,  $K$ ,  $IV$   
 o 출력 :  $P=\{P_1, \dots, P_n\}$   
 o 처리과정  
 $C_0 = IV$   
 for  $i=1$  to  $n$   
 $I_i = C_i$   
 $O_i = D_K(I_i)$   
 $P_i = C_{i-1} \oplus O_i$

< CBC 복호화 의사코드 >



< CBC 복호화 >

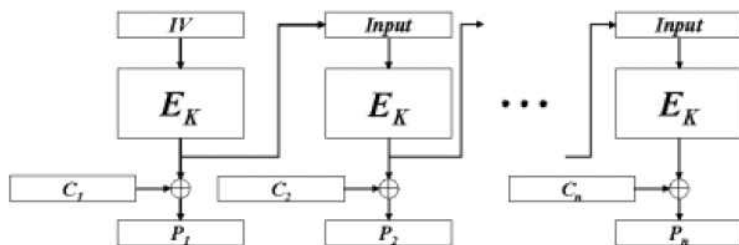
### 3.3.3 CTR 운영모드

CTR(CounTeR) 모드는 각 단계에 따라 증가되는 카운트 블록을 입력 블록으로 사용하는 운영모드이다.

CTR 모드의 암호화 과정은 카운터( $ctr+(i-1)$ )를 현 단계의 입력 블록( $I_i$ )으로 하여 암호화한 출력 블록( $O_i$ )을 평문 블록( $P_i$ )과 배타적 논리합 연산을 수행함으로 암호문 블록( $C_i$ )을 생성한다.

o 입력 :  $C=\{C_1, \dots, C_n\}$ ,  $K$ ,  $ctr$   
 o 출력 :  $P=\{P_1, \dots, P_n\}$   
 o 처리과정  
 for  $i=1$  to  $n$   
 $I_i = ctr+(i-1) \bmod 2^{128}$   
 $O_i = E_K(I_i)$   
 $P_i = C_i \oplus O_i$

< CTR 암호화 의사코드 >

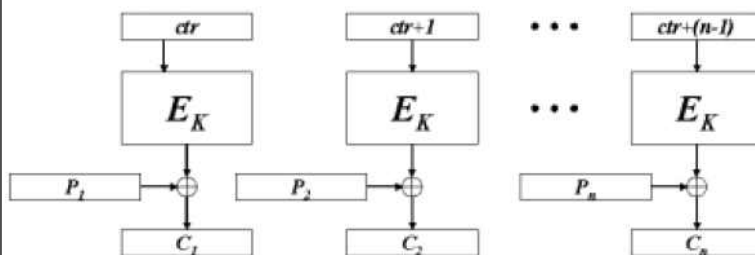


< CTR 암호화 >

CTR 모드의 복호화 과정은 카운터( $ctr+(i-1)$ )를 현 단계의 입력 블록( $I_i$ )으로 하여 암호화한 출력 블록( $O_i$ )을 암호문 블록( $C_i$ )과 배타적 논리합 연산을 수행함으로 평문 블록( $P_i$ )을 생성한다.

o 입력 :  $P=\{P_1, \dots, P_n\}$ ,  $K$ ,  $ctr$   
 o 출력 :  $C=\{C_1, \dots, C_n\}$   
 o 처리과정  
 for  $i=1$  to  $n$   
 $I_i = ctr+(i-1) \bmod 2^{128}$   
 $O_i = E_K(I_i)$   
 $C_i = P_i \oplus O_i$

< CTR 복호화 의사코드 >



< CTR 복호화 >

### 3.4. 패딩방법

메시지를 SEED에 입력하기 위해 여러 개의 128비트 블록으로 나눌 때, 마지막 블록을 정확히 128비트 블록의 크기로 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 입력값으로 사용할 수 있다. 이렇게 부족한 부분을 채우는 방식을 패딩이라고 한다.

ECB, CBC 모드는 평문 블록을 암호화의 입력으로 사용하기 때문에, 평문 데이터의 크기가 128비트의 양의 정수배가 되어야 하기 때문에, 반드시 패딩 방법이 적용되어야만 한다. 그러나, CFB-s, OFB, CTR 모드의 경우, 마지막 암호화 단계의 평문 블록이 128비트(CFB-s 모드의 경우 s비트)를 만족하지 못하고 m비트가 남아있을 때, 마지막 출력 블록(On) 128비트 중 상위 m비트 (MSBs(On))와 마지막 평문 블록 m비트와 배타적 논리합 연산함으로써 아래와 같은 패딩 방법을 적용하지 않을 수 있다.

본 패딩 방법은 ISO/IEC 국제표준 및 PKCS에서 사용되는 방법으로, 적용되는 시스템에 맞도록 사용함을 권고하나, '패딩 방법 1'은 복호화된 평문 데이터의 크기가 명확하지 않은 경우가 발생하기 때문에 평문 데이터의 크기가 명확히 알려져 있는 경우에 사용함을 권고한다.

#### 3.4.1 패딩 방법 1

평문 데이터의 크기가 128비트 양의 정수배가 아닐 때, 마지막 평문 블록이 128비트가 되도록 바이트 '00'을 덧붙인다.

```
예) 입력 블록( 48비트) : 4F 52 49 54 48 4D
    패딩 블록(128비트) : 4F 52 49 54 48 4D 00 00 00 00 00 00 00 00 00 00

    입력 블록(128비트) : 53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78
    패딩 블록(128비트) : 53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78
```

#### 3.4.2 패딩 방법 2

평문 데이터의 크기가 128비트의 양의 정수배가 아닐 경우, 마지막 평문 블록이 128비트가 되도록 바이트 '80' 덧붙인 후 나머지에 모두 '00' 바이트를 덧붙이고, 평문 블록의 크기가 128비트의 양의 정수배일 경우, 추가적인 128비트 '80...00' 블록을 추가한다.

```
예) 입력 블록( 48비트) : 4F 52 49 54 48 4D
    패딩 블록(128비트) : 4F 52 49 54 48 4D 80 00 00 00 00 00 00 00 00 00

    입력 블록(128비트) : 53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78
    패딩 블록(256비트) : 53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78
                        80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

### 3.4.3 패딩 방법 3

평문 데이터의 크기가 128비트의 양의 정수배가 아닐 경우, 마지막 평문 블록이 128비트가 되도록 덧붙이기가 필요한 바이트 수 'xx...xx'를 덧붙이고, 평문 블록의 크기가 128비트의 양의 정수배일 경우, 추가적인 128비트 '0F...0F' 블록을 추가한다.

예) 입력 블록( 48비트) : 4F 52 49 54 48 4D  
 패딩 블록(128비트) : 4F 52 49 54 48 4D 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A

입력 블록(128비트) : 53 45 45 44 41 4C 47 A8 3E D1 AF 07 4A 73 12 2C  
 패딩 블록(256비트) : 53 45 45 44 41 4C 47 A8 3E D1 AF 07 4A 73 12 2C  
 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F

### 3.5. 활용 방법

SEED 알고리즘은 16Byte의 비밀키, 16Byte의 평문을 입력으로 16Byte의 암호문을 출력하는 블록 암호 알고리즘이다.

입력	출력	키 길이	라운드
128 bits (16 bytes)	128 bits (16 bytes)	128 bits (16 bytes)	16 라운드

소스 코드에는 2가지 방법의 함수가 포함되어 있다.

**방법 1**은 처리하고자 하는 데이터가 적을 경우이며 Encrypt와 Decrypt 함수만 호출하면 암호화와 복호화가 된다.

**방법 2**는 대용량의 데이터를 암호화/복호화 할 때 Initialize, Process, Close 3가지 단계로 데이터 크기가 버퍼보다 클 경우 사용되는 방법이다.

## 4. 응용 프로그램

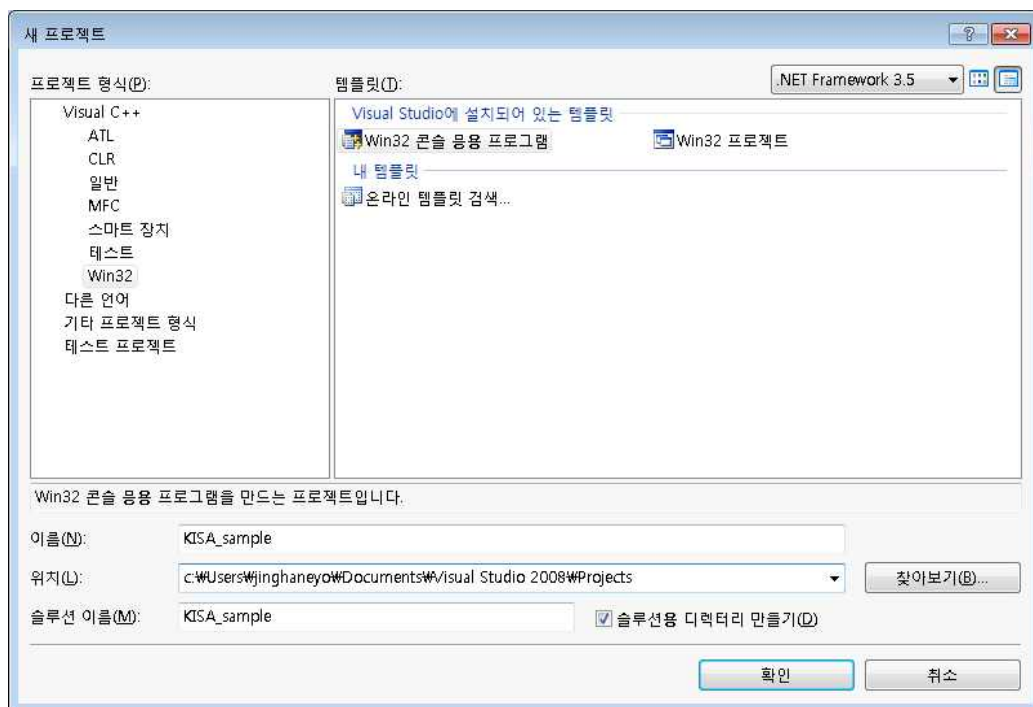
Windows, Linux 등 다양한 운영체제 환경에서 응용 프로그램을 개발과 함께 암호화를 적용할 수 있도록 국산암호 소스코드를 개발하였습니다. 기본적으로 많이 사용하는 C/C++ 및 Java를 기반으로 하며 이번 소스코드에 포함된 운영모드는 ECB, CBC, CTR이 함께 제공됩니다.

### 4.1. C/C++

프로젝트 생성 및 소스 추가, 빌드 등 배포되는 소스코드를 이용하여 암호화/복호화를 실행하는 방법에 대해서는 Microsoft 社의 Visual studio 2008을 활용하여 설명하도록 한다.

#### 4.1.1 프로젝트 생성 및 빌드

Visual studio를 실행하여 “Win32 콘솔 응용 프로그램”을 선택한다. 이름에는 프로젝트 명을 기입하고 위치에는 생성시키고자 하는 곳의 위치를 지정하여 준다.

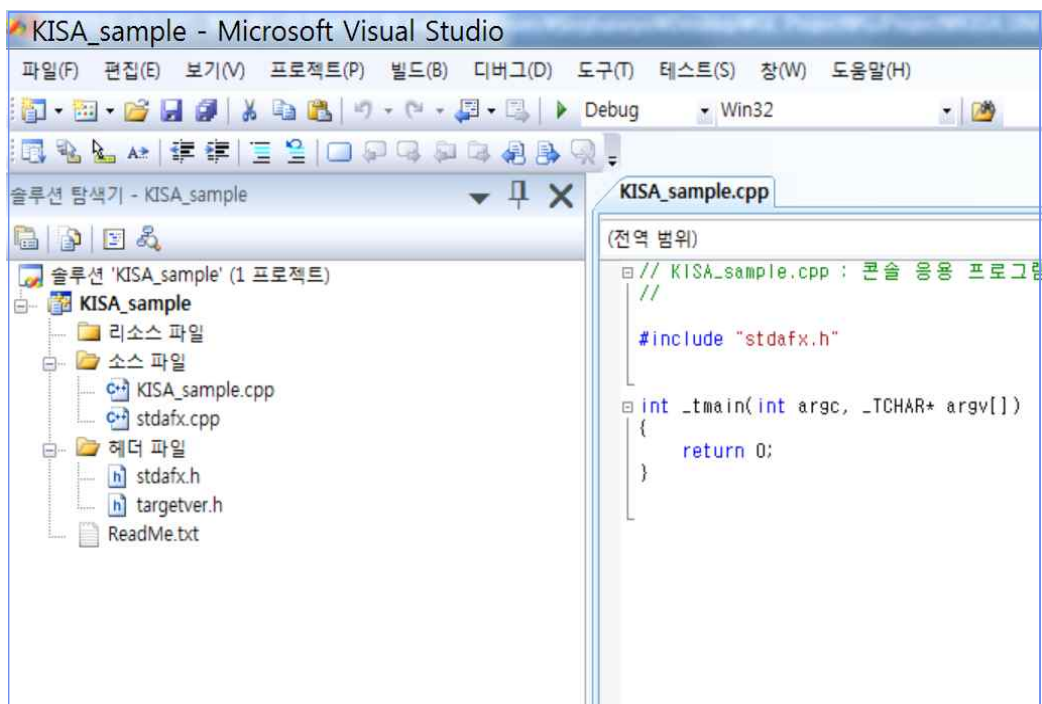


“콘솔 응용 프로그램”으로 선택한 후 마침 버튼을 누르면 콘솔 형태의 프로젝트가 생성된다. 미리 컴파일된 헤더를 체크해주면 `int _tmain( int argc, TCHAR *argv[] )`을 자동으로 생성 시켜 준다.

빈 프로젝트는 말그대로 아무것도 없는 것으로 직접 헤더 파일과 소스파일을 추가하여 전부 작성하는 것이다. 여기서는 미리 컴파일된 헤더를 사용하여 프로젝트를 구성하도록 한다.

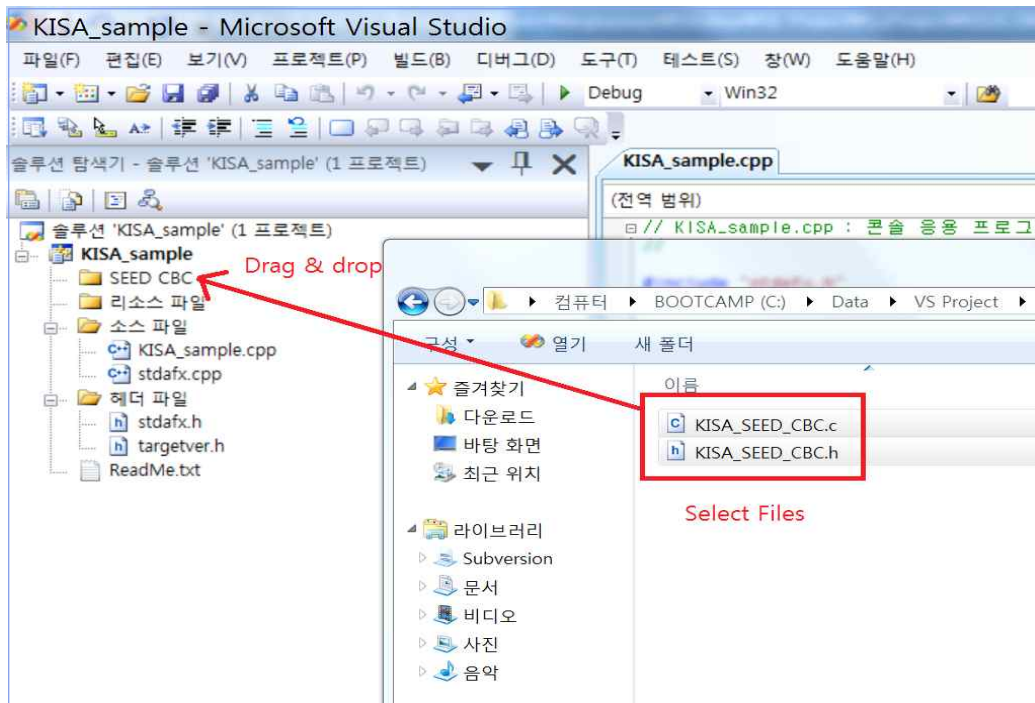


프로젝트 생성이 완료되면 “콘솔 응용 프로그램“을 작성할 수 있도록 기본적인 .h와 .cpp의 파일들을 생성된다. KISA\_sample.cpp의 \_tmain 함수 내에서 암호화/복호화 소스를 활용하도록 한다.

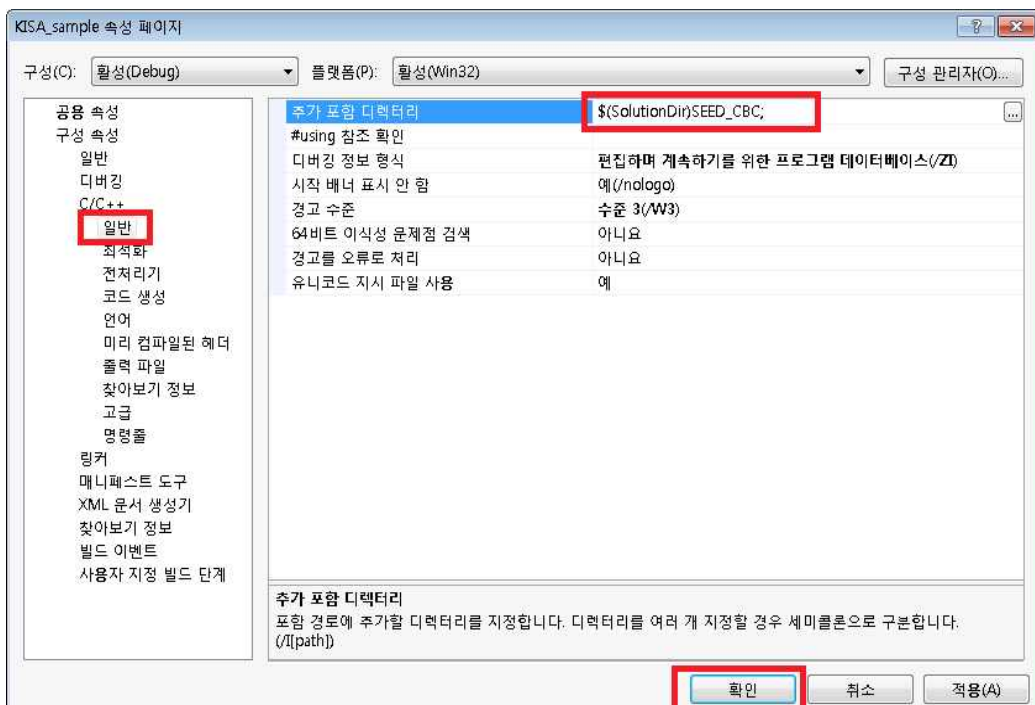


프로젝트 생성이 완료된 후, KISA\_sample 프로젝트 하위로 “SEED xxx“ 폴더를 추가하도록 한다. KISA에서 배포하는 소스코드 ”KISA\_SEED\_xxx.c“ 및 ”KISA\_SEED\_xxx.h“ 파일을 드래그 앤 드롭하여

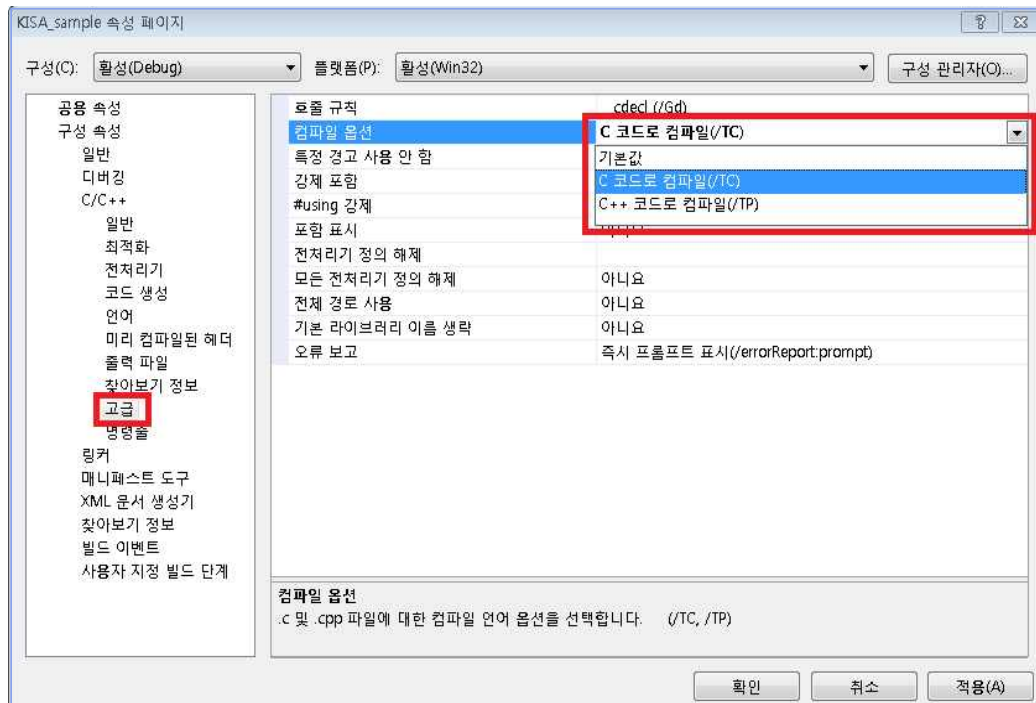
해당 폴더로 복사한다.



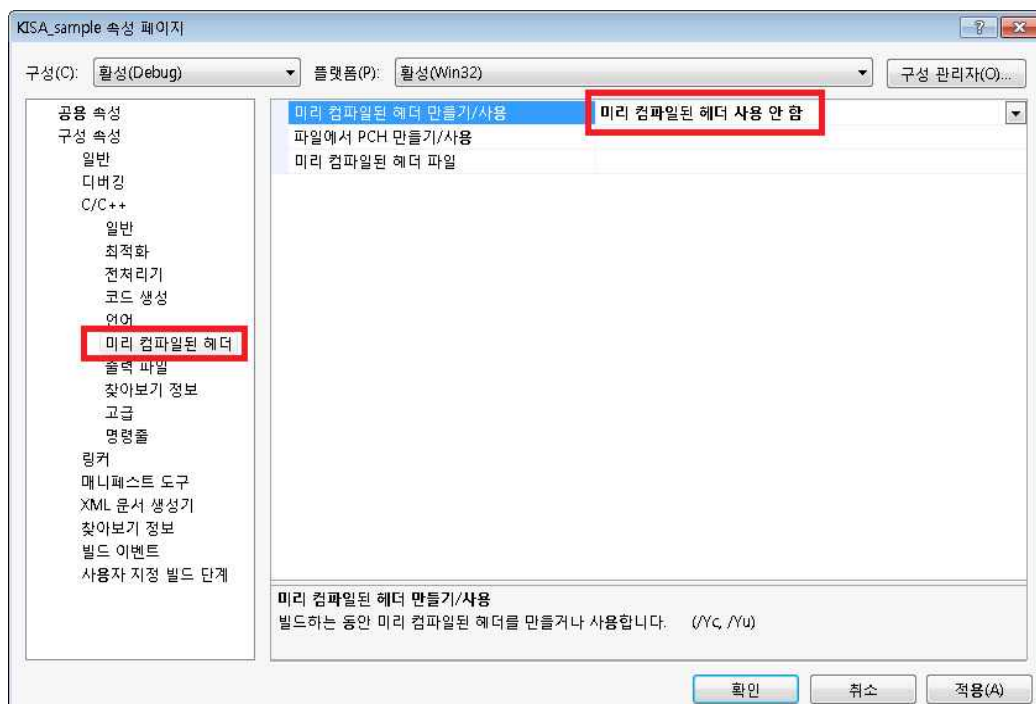
프로젝트 속성에서 C/C++의 추가 포함 디렉터리 항목에 포함 시킨 헤더와 소스 파일이 있는 폴더의 절대 경로 또는 상대 경로를 포함시켜야 한다. Visual studio에서 제공되는 매크로 함수를 이용한 상대 경로를 추가하였다. 매크로 관련해서는 도움말을 이용하고 여기서는 따로 설명을 하지 않는다. 여러 개의 폴더 또한 포함 시킬 수 있으며 구분자는 ";"로 하여 연속하여 기입 하면 된다.



또한, 고급에서 컴파일러 옵션을 "C 코드로 컴파일(/IC)"로 변경한다. C++로 해도 컴파일이 가능하나 배포되는 소스코드가 C이므로 C컴파일러로 구성하도록 한다.



마지막으로 .c 파일의 경우 “미리 컴파일된 헤더 사용 안 함”으로 설정하면 초기 빌드 할 수 있는 환경이 다 갖추어 졌다.





#### 4.1.2 소스코드 설명

##### o SEED-ECB

##### ☞ 함수 설명

```
void SEED_KeySched( DWORD *pdwRoundKey, BYTE *UserKey );
```

SEED-ECB Round Key 생성 함수

##### 매개변수 :

pdwRoundKey    생성될 라운드키 버퍼

pbUserKey      사용자가 지정하는 입력 키

##### 반환값 :

- 없음

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

SEED-ECB 알고리즘 암호화 함수

##### 매개변수 :

pbData            사용자 입력 평문

pdwRoundKey    암호화에 사용할 라운드 키

##### 반환값 : 없음

##### 참 고 :

1. 암호화 이전에 반드시 라운드키를 생성해야 한다.

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

SEED-ECB 알고리즘 복호화 함수

##### 매개변수 :

pbData            사용자 입력 암호문

pdwRoundKey    복호화에 사용할 라운드 키

##### 반환값 : 없음

##### 참 고 :

1. 암호화 이전에 반드시 라운드키를 생성해야 한다.

☞ 테스트 페이지

```

C:\windows\system32\cmd.exe

Key       : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Plaintext  : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Encryption....
Ciphertext : 5E BA C6 E0 05 4E 16 68 19 AF F1 CC 6D 34 6C DB

Decryption....
Plaintext  : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Round Key :
K 1,0 : 7C8F8C7E      K 1,1 : C737A22C
K 2,0 : FF276CDB      K 2,1 : A7CA684A
K 3,0 : 2F9D01A1      K 3,1 : 70049E41
K 4,0 : AE59B3C4      K 4,1 : 4245E90C
K 5,0 : A1D6400F      K 5,1 : DBC1394E
K 6,0 : 85963508      K 6,1 : 0C5F1FCB
K 7,0 : B684BDA7      K 7,1 : 61A4AEAE
K 8,0 : D17E0741      K 8,1 : FEE90AA1
K 9,0 : 76CC05D5      K 9,1 : E97A7394
K10,0 : 50AC6F92      K10,1 : 1B2666E5
K11,0 : 65B7904A      K11,1 : 8EC3A7B3
K12,0 : 2F7E2E22      K12,1 : A2B121B9
K13,0 : 4D0BFDE4      K13,1 : 4E888D9B
K14,0 : 631C8DDC      K14,1 : 4378A6C4
K15,0 : 216AF65F      K15,1 : 7878C031
K16,0 : 71891150      K16,1 : 98B255B0
  
```

## o SEED-CBC

### ☞ 함수 설명

```
int SEED_CBC_Encrypt( IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE
*pbszPlainText, IN int nPlainTextLen, OUT BYTE *pbszCipherText )
```

SEED-CBC 알고리즘 암호화 함수(처리하고자 하는 데이터가 적을 경우에 사용)

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키(16 BYTE)
pszbIV	사용자가 지정하는 초기화 벡터(16 BYTE)
pbszPlainText	사용자 입력 평문
nPlainTextLen	평문 길이(BYTE 단위의 평문길이)
pbszCipherText	암호문 출력 버퍼

#### 반환값 :

암호문의 Byte 길이

#### 참 고 :

패딩 로직때문에 16바이트 블록으로 처리함으로 pbszCipherText는 평문보다 16바이트 커야 한다.  
(평문이 16바이트 블록 시 패딩 데이터가 16바이트가 들어간다.)

```
int SEED_CBC_Decrypt( IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE
*pbszCipherText, IN int nCipherTextLen, OUT BYTE *pbszPlainText )
```

SEED-CBC 알고리즘 복호화 함수(처리하고자 하는 데이터가 적을 경우에 사용)

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키(16 BYTE)
pszbIV	사용자가 지정하는 초기화 벡터(16 BYTE)
pbszCipherText	암호문
nCipherTextLen	암호문 길이(BYTE 단위의 평문길이)
pbszPlainText	평문 출력 버퍼

#### 반환값 :

평문의 Byte 길이

```
int SEED_CBC_init( OUT KISA_SEED_INFO *pInfo, IN KISA_ENC_DEC enc, IN BYTE
*pbszUserKey, IN BYTE *pbszIV )
```

SEED-CBC 알고리즘 초기화 함수

**매개변수 :**

pInfo            CBC 내부에서 사용되는 구조체로써 유저가 변경하면 안된다.  
                  (메모리 할당되어 있어야 한다.)

enc              암호화 및 복호화 모드 지정  
                  (암호화 : KISA\_ENCRYPT / 복호화 : KISA\_DECRYPT)

pbszUserKey    사용자가 지정하는 입력 키(16 BYTE)

pbszIV          사용자가 지정하는 초기화 벡터(16 BYTE)

**반환값 :**

- 0: pInfo 또는 pbszUserKey 또는 pbszIV 포인터가 NULL일 경우
- 1: 초기화성공

```
int SEED_CBC_Process( OUT KISA_SEED_INFO *pInfo, IN DWORD *in, IN int inLen, OUT
DWORD *out, OUT int *outLen )
```

SEED-CBC 다중 블록 암호화/복호화 함수

**매개변수 :**

pInfo            SEED\_CBC\_init 에서 설정된 KISA\_HIGHT\_INFO 구조체

in                평문/암호문 (평문은 chartoint32\_for\_SEED\_CBC를 사용하여 int로 변환된 데이터)

inLen            평문/암호문 길이(BYTE 단위)

out              평문/암호문 버퍼

outLen           진행된 평문/암호문의 길이(BYTE 단위로 넘겨준다)

**반환값 :**

- 0: inLen의 값이 0보다 작은 경우,  
          KISA\_SEED\_INFO 구조체나 in, out에 널 포인터가 할당되었을 경우
- 1: 성공

```
int SEED_CBC_Close( OUT KISA_SEED_INFO *pInfo, IN DWORD *out, IN int *outLen )
```

SEED-CBC 운영모드 종료 및 패딩 처리(PKCS7)

**매개변수 :**

pInfo            SEED\_CBC\_Process 를 거친 KISA\_HIGHT\_INFO 구조체  
out              평문/암호문 출력 버퍼  
outLen           출력 버퍼에 저장된 데이터 길이(BYTE 단위의 평문길이)

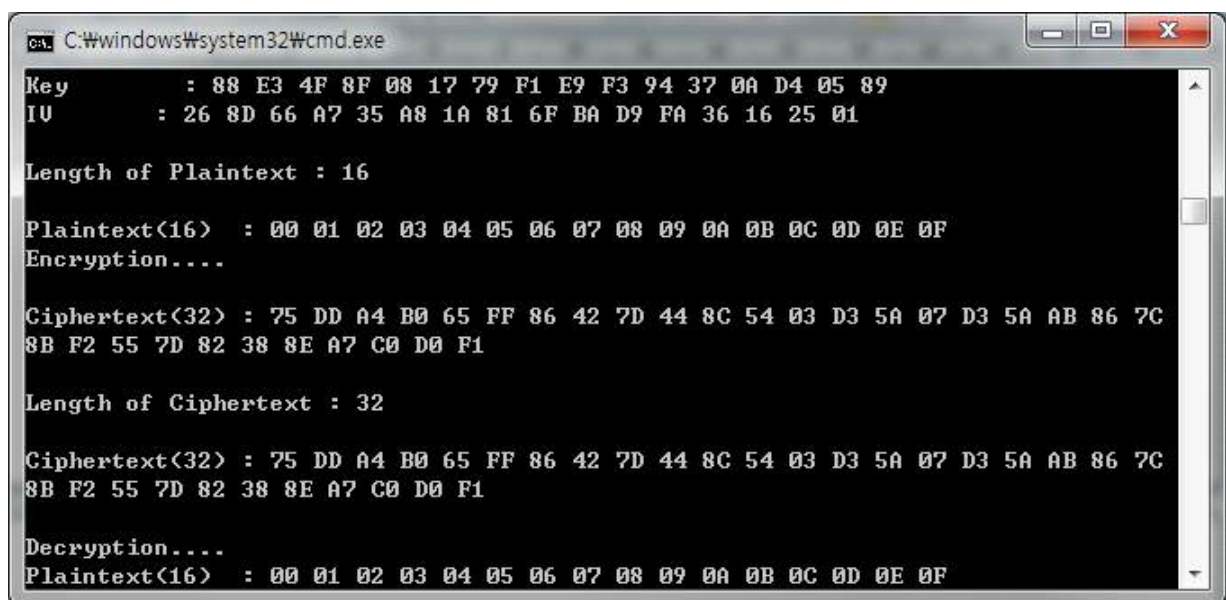
**반환값 :**

- 0 :    inLen의 값이 0보다 작은 경우,  
         KISA\_SEED\_INFO 구조체나 out에 널 포인터가 할당되었을 경우
- 1 :    성공

**참 고 :**

패딩 로직때문에 16바이트 블록으로 처리함으로 복호화 시 출력 버퍼는 평문보다 16바이트 커야 한다.(평문이 16바이트 블록 시 패딩 데이터가 16바이트가 들어간다.)

☞ 테스트 페이지



```
C:\windows\system32\cmd.exe

Key       : 88 E3 4F 8F 08 17 79 F1 E9 F3 94 37 0A D4 05 89
IV        : 26 8D 66 A7 35 A8 1A 81 6F BA D9 FA 36 16 25 01

Length of Plaintext : 16

Plaintext<16> : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Encryption....

Ciphertext<32> : 75 DD A4 B0 65 FF 86 42 7D 44 8C 54 03 D3 5A 07 D3 5A AB 86 7C
8B F2 55 7D 82 38 8E A7 C0 D0 F1

Length of Ciphertext : 32

Ciphertext<32> : 75 DD A4 B0 65 FF 86 42 7D 44 8C 54 03 D3 5A 07 D3 5A AB 86 7C
8B F2 55 7D 82 38 8E A7 C0 D0 F1

Decryption....
Plaintext<16> : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
```

## o SEED-CTR

### ☞ 함수 설명

```
int SEED_CTR_Encrypt( IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE
*pbszPlainText, IN int nPlainTextLen, OUT BYTE *pbszCipherText )
```

SEED-CTR 알고리즘 암호화 함수(처리하고자 하는 데이터가 적을 경우에 사용)

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키(16 BYTE)
pszbIV	사용자가 지정하는 초기화 벡터(16 BYTE)
pbszPlainText	사용자 입력 평문
nPlainTextLen	평문 길이(BYTE 단위의 평문길이)
pbszCipherText	암호문 출력 버퍼

#### 반환값 :

암호문의 Byte 길이

#### 참 고 :

패딩 로직때문에 16바이트 블록으로 처리함으로 pbszCipherText는 평문보다 16바이트 커야 한다.  
(평문이 16바이트 블록 시 패딩 데이터가 16바이트가 들어간다.)

```
int SEED_CTR_Decrypt( IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE
*pbszCipherText, IN int nCipherTextLen, OUT BYTE *pbszPlainText )
```

SEED-CTR 알고리즘 복호화 함수(처리하고자 하는 데이터가 적을 경우에 사용)

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키(16 BYTE)
pszbIV	사용자가 지정하는 초기화 벡터(16 BYTE)
pbszCipherText	암호문
nCipherTextLen	암호문 길이(BYTE 단위의 평문길이)
pbszPlainText	평문 출력 버퍼

#### 반환값 :

평문의 Byte 길이

```
int SEED_CTR_init( OUT KISA_SEED_INFO *pInfo, IN KISA_ENC_DEC enc, IN BYTE
*pszUserKey, IN BYTE *pbszCounter )
```

SEED-CTR 알고리즘 초기화 함수

**매개변수 :**

pInfo                CTR 내부에서 사용되는 구조체로써 유저가 변경하면 안된다.  
                       (메모리 할당되어 있어야 한다.)

enc                  암호화 및 복호화 모드 지정

pbszUserKey        사용자가 지정하는 입력 키(16 BYTE)

pbszIV              사용자가 지정하는 초기화 벡터(16 BYTE)

**반환값 :**

- 0: pInfo 또는 pbszUserKey 또는 pbszIV 포인터가 NULL일 경우
- 1: 초기화성공

```
int SEED_CTR_Process( OUT KISA_SEED_INFO *pInfo, IN DWORD *in, IN int inLen, OUT
DWORD *out, OUT int *outLen )
```

SEED CTR 다중 블록 암호화/복호화 함수

**매개변수 :**

pInfo                SEED\_CTR\_init 에서 설정된 KISA\_HIGHT\_INFO 구조체

in                    평문/암호문 (평문은 chartoint32\_for\_SEED\_CTR를 사용하여 int로 변환된 데이터)

inLen                평문/암호문 길이(BYTE 단위)

out                  평문/암호문 버퍼

outLen              진행된 평문/암호문의 길이(BYTE 단위로 넘겨준다)

**반환값 :**

- 0: inLen의 값이 0보다 작은 경우,  
       KISA\_SEED\_INFO 구조체나 in, out에 널 포인터가 할당되었을 경우
- 1: 성공

```
int SEED_CTR_Close( OUT KISA_SEED_INFO *pInfo, IN DWORD *out, IN int *outLen )
```

SEED-CTR 운영모드 종료 및 패딩 처리(PKCS7)

**매개변수 :**

pInfo            SEED\_CTR\_Process 를 거친 KISA\_HIGHT\_INFO 구조체  
out              평문/암호문 출력 버퍼  
outLen           출력 버퍼에 저장된 데이터 길이(BYTE 단위의 평문길이)

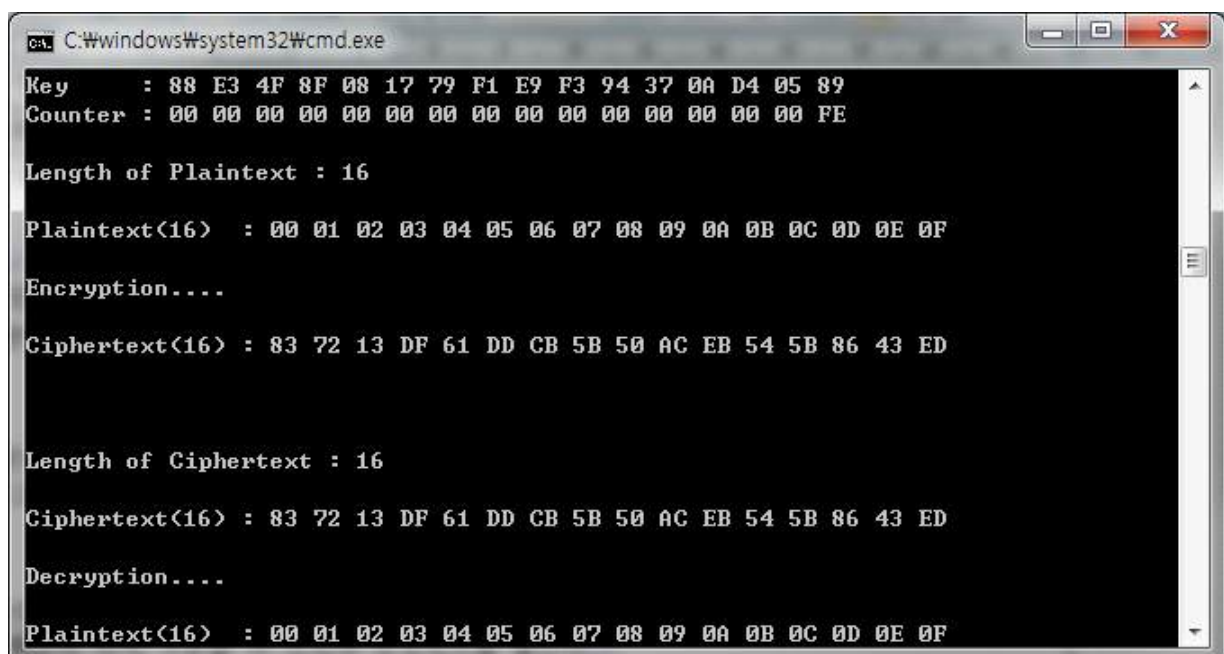
**반환값 :**

- 0 :    inLen의 값이 0보다 작은 경우,  
         KISA\_SEED\_INFO 구조체나 out에 널 포인터가 할당되었을 경우
- 1 :    성공

**참 고 :**

패딩 로직때문에 16바이트 블록으로 처리함으로 복호화 시 출력 버퍼는 평문보다 16바이트 커야 한다.(평문이 16바이트 블록 시 패딩 데이터가 16바이트가 들어간다.)

☞ 테스트 페이지



```
C:\windows\system32\cmd.exe
Key       : 88 E3 4F 8F 08 17 79 F1 E9 F3 94 37 0A D4 05 89
Counter   : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE

Length of Plaintext : 16

Plaintext<16> : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Encryption....

Ciphertext<16> : 83 72 13 DF 61 DD CB 5B 50 AC EB 54 5B 86 43 ED

Length of Ciphertext : 16

Ciphertext<16> : 83 72 13 DF 61 DD CB 5B 50 AC EB 54 5B 86 43 ED

Decryption....

Plaintext<16> : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
```

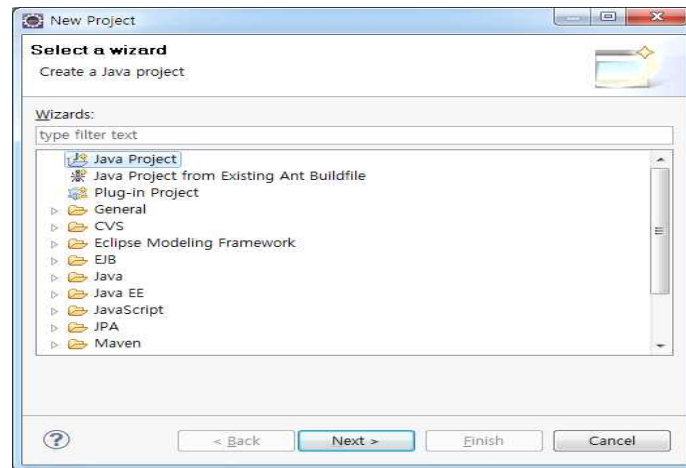


## 4.2. Java

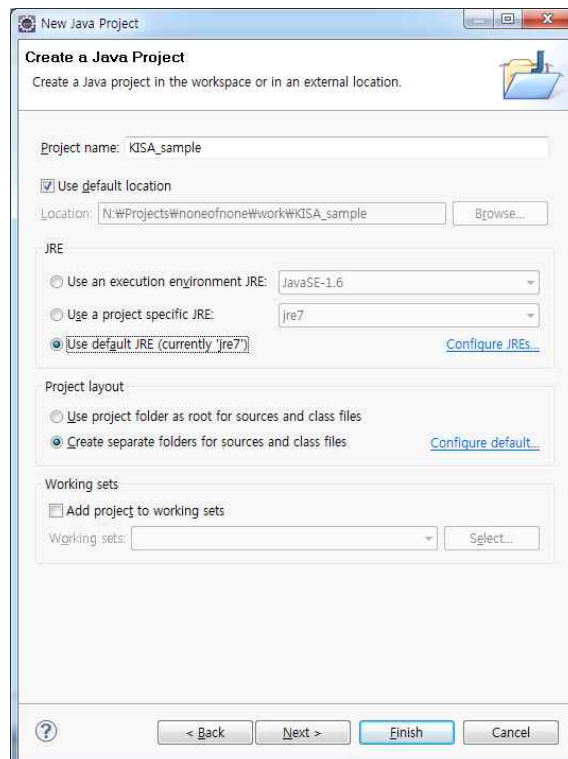
다음은 Java 형태의 프로그램을 하나 생성하여 헤더 및 소스와 파일 추가 하는 방법과 빌드하는 방법 등을 설명한다.

### 4.2.1 프로젝트 생성 및 빌드

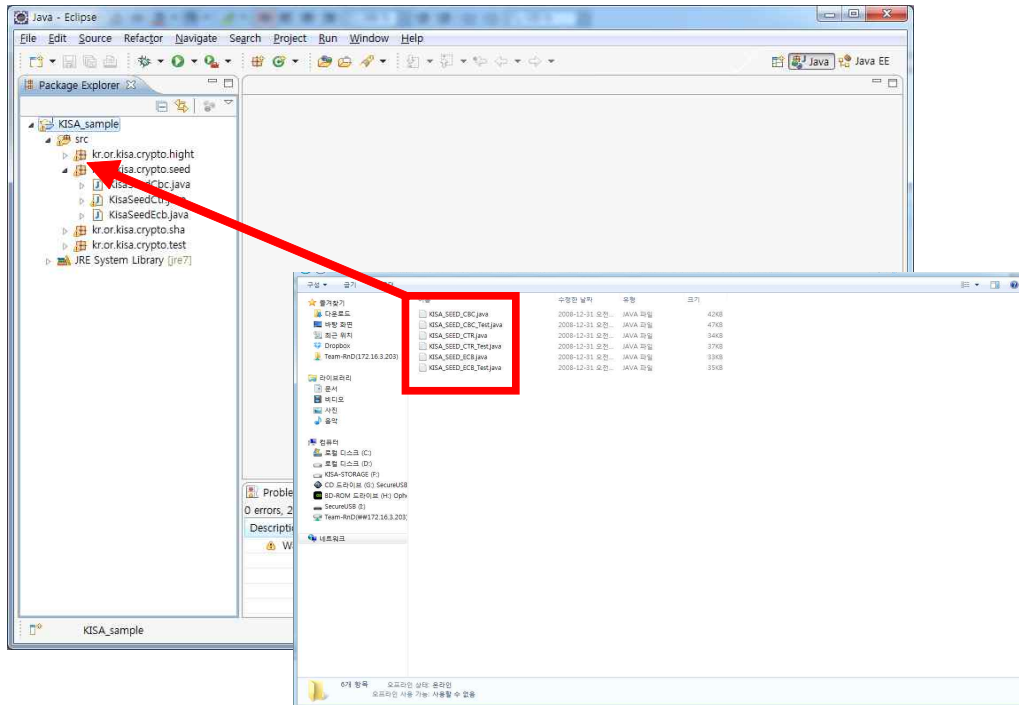
Eclipse를 실행하여 하기 이미지처럼 Java 프로젝트를 선택한다.



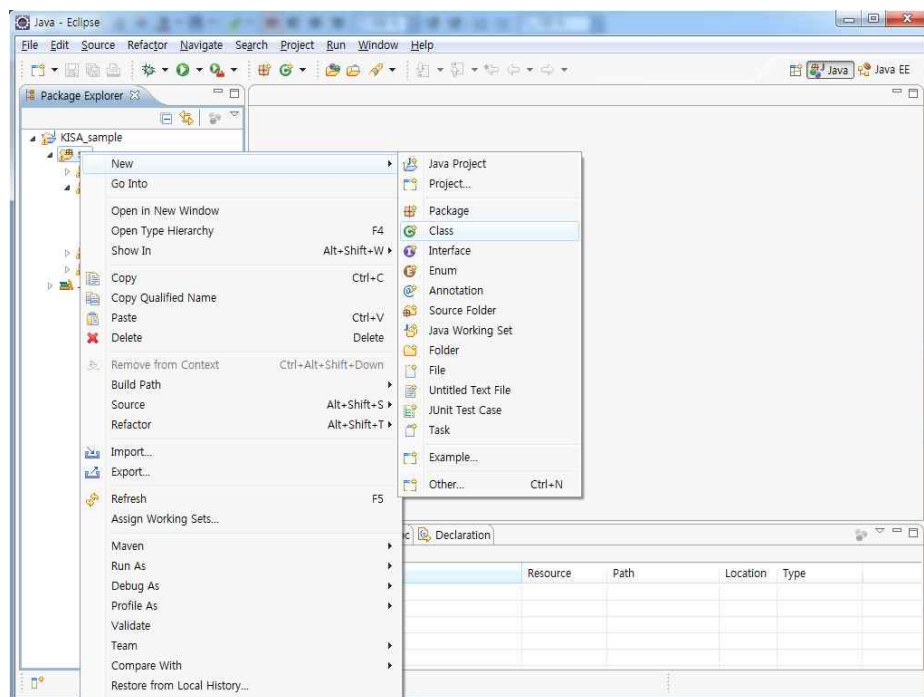
Next를 클릭하여 Project name에는 프로젝트 명을 기입한다. "Use default JRE" 를 선택 후 Finish 버튼을 눌러 프로젝트 생성을 완료한다.



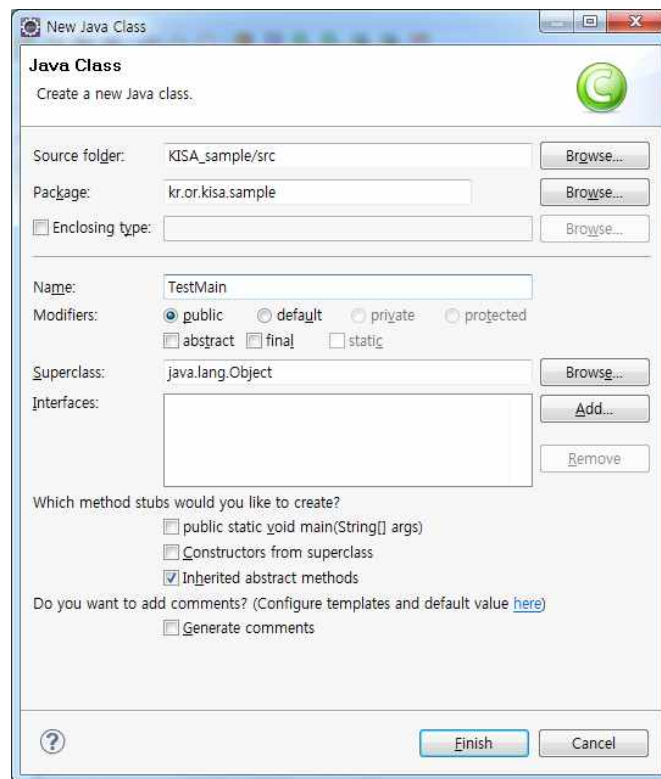
소스가 있는 폴더로 이동하여 프로젝트에 추가할 파일을 드래그&드롭으로 프로젝트로 이동시킨다. 이때 “Copy files and folders”를 선택하여 복사하여 준다.



테스트 클래스를 생성하여 배포 중인 소스를 사용할 수 있도록 구성한다. 먼저, src 폴더를 마우스로 우 클릭하여 New -> Class를 선택한다.

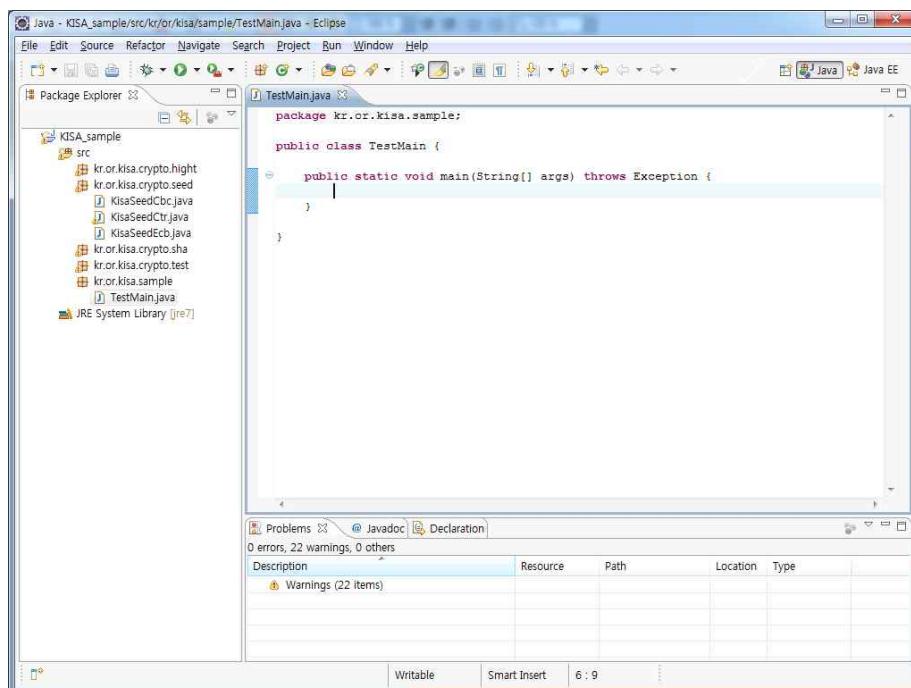


다음으로는 Package와 Name을 입력 후 Finish 버튼을 클릭하여 테스트 클래스를 생성한다.



생성된 빈 클래스 안에 아래와 같이 메인 함수를 추가 후 작업을 하면 된다.

```
public static void main(String[] args) throws Exception {
}
```



#### 4.2.2 소스코드 설명

##### o SEED-ECB

###### 함수 설명

```
public static byte[] SEED_ECB_Encrypt( byte[] pbszUserKey, byte[] pbData, int offset, int length );
```

SEED-ECB 알고리즘 암호화 함수

###### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbData	사용자 입력 평문
offset	사용자 입력 길이 시작 오프셋
length	사용자 입력 길이

###### 반환값 :

- 사용자 입력에 대한 암호문 출력 byte

###### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

```
public static byte[] SEED_ECB_Decrypt( byte[] pbszUserKey, byte[] pbData, int offset, int length );
```

SEED-ECB 알고리즘 복호화 함수

###### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbData	사용자 입력 암호문
offset	사용자 입력 길이 시작 오프셋
length	사용자 입력 길이

###### 반환값 :

- 사용자 입력에 대한 평문 출력 byte

###### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

## 🔗 테스트 페이지

[ Test SEED reference code ECB ]

Key : 88 e3 4f 8f 8 17 79 f1 e9 f3 94 37 a d4 5 89  
Plaintext : d7 6d d 18 32 7e c5 62 b1 5e 6b c3 65 ac c f

Ciphertext(SEED\_ECB\_Encrypt) : f 4e 7f c7 8c 48 d5 ad 95 10 ba d8 98 7b a5 22 99 9 af 8b b8 17 43 42 7 6c 86 53 e9 e5 cf b8  
Plaintext(SEED\_ECB\_Decrypt) : d7 6d d 18 32 7e c5 62 b1 5e 6b c3 65 ac c f

o SEED-CBC

```
public static byte[] SEED_CBC_Encrypt( byte[] pbszUserKey, byte[] pbszIV, byte[] message, int message_offset, int message_length )
```

SEED-CBC 알고리즘 암호화 함수

**매개변수 :**

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

**반환값 :**

- 사용자 입력에 대한 암호문 출력 byte

**참 고 :**

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 한다.

```
public static byte[] SEED_CBC_Decrypt( byte[] pbszUserKey, byte[] pbszIV, byte[] message, int message_offset, int message_length )
```

SEED-CBC 알고리즘 복호화 함수

**매개변수 :**

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

**반환값 :**

- 사용자 입력에 대한 평문 출력 byte

**참 고 :**

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 한다.

```
public static int SEED_CBC_init( KISA_SEED_INFO pInfo, KISA_ENC_DEC enc, byte[]
pbszUserKey, byte[] pbszIV )
```

SEED-CBC 알고리즘 초기화 함수

**매개변수 :**

pInfo            SEED CBC 알고리즘 운영을 위한 클래스 지정  
enc                알고리즘 암호화 및 복호화 모드 지정  
                  (암호화 : KISA\_ENCRYPT / 복호화 : KISA\_DECRYPT)  
pbszUserKey      사용자가 지정하는 입력 키 (16 bytes)  
pbszIV            사용자가 지정하는 초기화 벡터 (16 bytes)

**반환값 :**

- 1 : 초기화 성공
- 0 : 초기화 실패

```
public static int SEED_CBC_Process( KISA_SEED_INFO pInfo, int[] in, int inLen, int[] out,
int[] outLen )
```

SEED-CBC 알고리즘 다중 블록 암호화 함수

**매개변수 :**

pInfo            SEED CBC 알고리즘 운영을 위한 클래스 지정  
                  (SEED\_CBC\_init으로 초기화 필요)  
in                사용자 입력 평문/암호문  
inLen            사용자 입력의 길이 지정  
out              사용자 입력에 대한 암호문/평문 출력 버퍼  
outLen           출력 버퍼에 저장된 데이터의 길이

**반환값 :**

- 1 : 구동 성공
- 0 : 구동 실패

```
public static int SEED_CBC_Close( KISA_SEED_INFO pInfo, int[] out, int out_offset, int[] outLen )
```

SEED-CBC 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

**매개변수 :**

pInfo	SEED CBC 알고리즘 운영을 위한 클래스 지정 (SEED_CBC_init으로 초기화 필요)
out	사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼
out_offset	출력 버퍼의 시작 오프셋
outLen	출력 버퍼에 저장된 데이터의 길이

**반환값 :**

- 1 : 패딩 성공
- 0 : 패딩 실패

## 테스트 페이지

[ Test SEED reference code CBC]

Key	: 88 e3 4f 8f 8 17 79 f1 e9 f3 94 37 a d4 5 89
Plaintext	: d7 6d d 18 32 7e c5 62 b1 5e 6b c3 65 ac c f
IV	: 26 8d 66 a7 35 a8 1a 81 6f ba d9 fa 36 16 25 1
Ciphertext(SEED_CBC_Encrypt)	: a2 93 ea e9 d9 ae bf ac 37 ba 71 4b d7 74 e4 27 4e 5c d7 b3 97 15 50 26 32 46 6c ab 29 ca ff 18
Plaintext(SEED_CBC_Decrypt)	: d7 6d d 18 32 7e c5 62 b1 5e 6b c3 65 ac c f



## o SEED-CTR

```
public static byte[] SEED_CTR_Encrypt( byte[] pbszUserKey, byte[] pbszCTR, byte[] message, int message_offset, int message_length )
```

SEED-CTR 알고리즘 암호화 함수

### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

### 반환값 :

- 사용자 입력에 대한 암호문 출력 byte

### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszCTR 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼의 크기와 동일( 패딩 처리를 하지 않는다. )

```
public static byte[] SEED_CTR_Decrypt( byte[] pbszUserKey, byte[] pbszCTR, byte[] message, int message_offset, int message_length )
```

SEED-CTR 알고리즘 복호화 함수

### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 암호문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

### 반환값 :

- 사용자 입력에 대한 평문 출력 byte

### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszCTR 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼 크기와 동일하다.

```
public static int SEED_CTR_init( KISA_SEED_INFO pInfo, KISA_ENC_DEC enc, byte[]
pszUserKey, byte[] pbszCTR )
```

SEED-CTR 알고리즘 초기화 함수

**매개변수 :**

pInfo                SEED CTR 알고리즘 운영을 위한 클래스 지정  
enc                    알고리즘 암호화 및 복호화 모드 지정  
                      (암호화 : KISA\_ENCRYPT / 복호화 : KISA\_DECRYPT)  
pbszUserKey        사용자가 지정하는 입력 키 (16 bytes)  
pbszCTR            사용자가 지정하는 초기 카운터 (16 bytes)

**반환값 :**

- 1 : 초기화 성공
- 0 : 초기화 실패

```
public static int SEED_CTR_Process( KISA_SEED_INFO pInfo, int[] in, int inLen, int[] out,
int[] outLen )
```

SEED-CTR 알고리즘 다중 블록 암호화 함수

**매개변수 :**

pInfo                SEED CTR 알고리즘 운영을 위한 클래스 지정  
                      (SEED\_CTR\_init으로 초기화 필요)  
in                    사용자 입력 평문/암호문  
inLen                사용자 입력의 길이 지정 (int 단위)  
out                   사용자 입력에 대한 암호문/평문 출력 버퍼  
outLen               출력 버퍼에 저장된 데이터의 길이 (int[] 단위)

**반환값 :**

- 1 : 구동 성공
- 0 : 구동 실패

**참 고 :**

1. 출력이 되는 버퍼의 크기는 사용자 입력의 길이보다 크거나 같게 미리 할당해야 함
2. outLen은 실제로 출력버퍼 out에 저장된 결과값의 길이를 함수 내부에서 지정함

```
public static int SEED_CTR_Close( KISA_SEED_INFO pInfo, int[] out, int out_offset, int[] outLen )
```

SEED-CTR 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

#### 매개변수 :

pInfo	SEED CTR 알고리즘 운영을 위한 클래스 지정 (SEED_CTR_init으로 초기화 필요)
out	사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼
out_offset	출력 버퍼의 시작 오프셋
outLen	출력 버퍼에 저장된 데이터의 길이

#### 반환값 :

- 1 : 패딩 성공
- 0 : 패딩 실패

#### ☞ 테스트 페이지

##### [ Test SEED reference code CTR]

Key	: 88 e3 4f 8f 8 17 79 f1 e9 f3 94 37 a d4 5 89
Plaintext	: d7 6d d 18 32 7e c5 62 b1 5e 6b c3 65 ac c f
CTR	: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 fe
Ciphertext(SEED_CTR_Encrypt)	: 54 1e 1c c4 57 a6 8 3e e9 fb 8a 9c 32 27 41 ed
Ciphertext(SEED_CTR_Decrypt)	: d7 6d d 18 32 7e c5 62 b1 5e 6b c3 65 ac c f

## 5. 웹 프로그램

Data Base, Homepage 등 다양한 Web Service 환경에서 전송·저장되는 구간의 암호화를 적용할 수 있도록 국산암호 소스코드를 개발하였다. 기본적으로 많이 사용하는 ASP, JSP, PHP를 기반으로 하며 이번 소스코드에 포함된 운영모드는 ECB, CBC, CTR이 함께 제공된다.

### 5.1. ASP

소스 활용 등 배포되는 소스코드를 이용하여 암호화/복호화를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, Windows Server 및 IIS 설정에 대한 설명은 생략하기로 한다.

#### 5.1.1 소스코드 추가

암호화/복호화 소스코드가 작성된 파일을 사용하고자 하는 파일에 포함 시킨다.

```
<!--#include file="KISA_SEED_ECB.asp" -->
<%response.charset = "utf-8" %>
<%
enc = Trim(request.form("ENC"))
dec = Trim(request.form("DEC"))
g_pbUserKey = Trim(request.form("KEY"))
Dim arrEnc
Dim arrDec
dim sampleData1
dim sampleData2
Dim enc2
Dim dec2
Dim arrKey
If(IsNull(g_pbUserKey) Or Len(Trim(g_pbUserKey))=0) Then
g_pbUserKey = "2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C"
End if

If(IsNull(enc) Or Len(Trim(enc))=0) Then
```

### 5.1.2 소스코드 설명

#### o SEED-ECB

##### ☞ 함수 설명

```
public function SEED_ECB_Encrypt(byref pbszUserKey, byref message, message_offset, message_length)
```

SEED-ECB 알고리즘 암호화 함수

##### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

##### 반환값 :

- 사용자 입력에 대한 암호문 출력 버퍼

##### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

```
public function SEED_ECB_Decrypt(byref pbszUserKey, byref message, message_offset, message_length)
```

SEED-ECB 알고리즘 복호화 함수

##### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
message	사용자 입력 암호문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

##### 반환값 :

- 사용자 입력에 대한 평문 출력 버퍼

##### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

## 국산 암호 [SEED-ECB] 테스트 페이지

<암호화 예제>		<복호화 예제>	
키(KEY):	2B, 7E, 15, 16, 28, AE, D2, A6, AB, F7, 15, 88, 09, CF, 4F, 3C	키(KEY):	2B, 7E, 15, 16, 28, AE, D2, A6, AB, F7, 15, 88, 09, CF, 4F, 3C
평문:	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F	암호문:	
	▼ 암호화		▼ 복호화
암호문:	F7, A5, AB, AA, 86, 9B, E1, 1E, C1, D0, 3B, BA, 92, 76, A1, 64, AD, 6C, 74, B7, 08, D1, CA, 7E, B1, AA, FF, 31, 96, 34, C6, 02	평문:	

\* 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex: 00,01,0A,0B)

<키(KEY)> : 2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C

<평문> : 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F

<암호문> : F7,A5,AB,AA,86,9B,E1,1E,C1,D0,3B,BA,92,76,A1,64,AD,6C,74,B7,08,D1,CA,7E,B1,AA,FF,31,96,34,C6,02

## o SEED-CBC

### 함수 설명

```
public function SEED_CBC_Encrypt( byref pbszUserKey, byref pbszIV, byref message, message_offset, message_length )
```

SEED-CBC 알고리즘 암호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

#### 반환값 :

- 사용자 입력에 대한 암호문 출력 버퍼

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 한다.

```
public function SEED_CBC_Decrypt(byref pbszUserKey, byref pbszIV, byref message, message_offset, message_length )
```

SEED-CBC 알고리즘 복호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 암호문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

#### 반환값 :

- 사용자 입력에 대한 평문 출력 버퍼

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 하다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 하다.

```
public function SEED_CBC_init(byref pInfo, enc, byref pbszUserKey, byref pbszIV )
```

SEED-CBC 알고리즘 초기화 함수

**매개변수 :**

pInfo	SEED CBC 알고리즘 운영을 위한 구조체 지정 (미리 메모리가 할당되어 있어야 함)
enc	알고리즘 암호화 및 복호화 모드 지정 (암호화 : KISA_ENCRYPT / 복호화 : KISA_DECRYPT)
pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)

**반환값 :**

- 1 : 초기화 성공
- 0 : 초기화 실패

```
public function SEED_CBC_Process( byref pInfo, byref v_in, inLen, byref v_out, byref outLen )
```

SEED-CBC 알고리즘 다중 블록 암호화 함수

**매개변수 :**

pInfo	SEED CBC 알고리즘 운영을 위한 구조체 지정 (KISA_SEED_CBC_init으로 초기화 필요)
v_in	사용자 입력 평문/암호문
inLen	사용자 입력의 길이 지정
v_out	사용자 입력에 대한 암호문/평문 출력 버퍼
outLen	출력 버퍼에 저장된 데이터의 길이

**반환값 :**

- 1 : 구동 성공
- 0 : 구동 실패



```
public function SEED_CBC_Close( byref pInfo, byref v_out, out_offset, byref outLen )
```

SEED-CBC 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

**매개변수 :**

pInfo SEED CBC 알고리즘 운영을 위한 구조체 지정  
(KISA\_SEED\_CBC\_init으로 초기화 필요)  
v\_out 사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼  
out\_offset 출력 버퍼의 시작 오프셋  
outLen 출력 버퍼에 저장된 데이터의 길이

**반환값 :**

- 1 : 패딩 성공
- 0 : 패딩 실패

☞ 테스트 페이지

## 국산 암호 [SEED-CBC] 테스트 페이지

<p>&lt;암호화 예제&gt;</p> <p>키(KEY): <input type="text" value="88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89"/></p> <p>초기값(IV): <input type="text" value="26,8D,66,A7,35,A8,1A,81,6F,BA,D9,FA,36,16,25,01"/></p> <p>평문: <input type="text" value="00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F"/></p> <p>▼ 암호화</p> <p>암호문: <input type="text" value="75,DD,A4,B0,65,FF,86,42,7D,44,8C,54,03,D3,5A,07,D3,5A,AB,86,7C,8B,F2,55,7D,82,38,8E,A7,C0,D0,F1"/></p>	<p>&lt;복호화 예제&gt;</p> <p>키(KEY): <input type="text" value="88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89"/></p> <p>초기값(IV): <input type="text" value="26,8D,66,A7,35,A8,1A,81,6F,BA,D9,FA,36,16,25,01"/></p> <p>암호문: <input type="text" value="75,DD,A4,B0,65,FF,86,42,7D,44,8C,54,03,D3,5A,07,D3,5A,AB,86,7C,8B,F2,55,7D,82,38,8E,A7,C0,D0,F1"/></p> <p>▼ 복호화</p> <p>평문: <input type="text" value="00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F"/></p>
--	--

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다 (ex : 00,01,0A,0B)

<키(KEY)> : 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89  
 <초기값(IV)> : 26,8D,66,A7,35,A8,1A,81,6F,BA,D9,FA,36,16,25,01  
 <평문> : 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F  
 <암호문> : 75,DD,A4,B0,65,FF,86,42,7D,44,8C,54,03,D3,5A,07,D3,5A,AB,86,7C,8B,F2,55,7D,82,38,8E,A7,C0,D0,F1

## o SEED-CTR

### 함수 설명

```
public function SEED_CTR_Encrypt( byref pbszUserKey, byref pbszCTR, byref message, message_offset, message_length )
```

SEED-CTR 알고리즘 암호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

#### 반환값 :

- 사용자 입력에 대한 암호문 출력 버퍼

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszCTR 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼의 크기와 동일( 패딩 처리를 하지 않는다. )

```
public function SEED_CTR_Decrypt(byref pbszUserKey, byref pbszCTR, byref message, message_offset, message_length)
```

SEED-CTR 알고리즘 복호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 암호문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

#### 반환값 :

- 사용자 입력에 대한 평문 출력 버퍼

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszCTR 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼 크기와 동일하다.

```
public function SEED_CTR_init(byref pInfo, enc, byref pbszUserKey, byref pbszCTR )
```

SEED-CTR 알고리즘 초기화 함수

**매개변수 :**

pInfo	SEED CTR 알고리즘 운영을 위한 구조체 지정 (미리 메모리가 할당되어 있어야 함)
enc	알고리즘 암호화 및 복호화 모드 지정 (암호화 : KISA_ENCRYPT / 복호화 : KISA_DECRYPT)
pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기 카운터(16 bytes)

**반환값 :**

- 1 : 초기화 성공
- 0 : 초기화 실패

```
public function SEED_CTR_Process( byref pInfo, byref v_in, inLen, byref v_out, byref outLen )
```

SEED-CTR 알고리즘 다중 블록 암호화 함수

**매개변수 :**

pInfo	SEED CTR 알고리즘 운영을 위한 구조체 지정 (KISA_SEED_init으로 초기화 필요)
v_in	사용자 입력 평문/암호문
inLen	사용자 입력의 길이 지정 (char 단위)
v_out	사용자 입력에 대한 암호문/평문 출력 버퍼
outLen	출력 버퍼에 저장된 데이터의 길이 (char 단위)

**반환값 :**

- 1 : 구동 성공
- 0 : 구동 실패

**참 고 :**

1. 출력이 되는 버퍼의 크기는 사용자 입력의 길이보다 크거나 같게 미리 할당 해야 함
2. outLen은 실제로 출력버퍼 v\_out에 저장된 결과값의 길이를 함수 내부에서 지정함

```
public function SEED_CTR_Close( byref pInfo, byref v_out, out_offset, byref outLen )
```

SEED-CTR 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

매개변수 :

pInfo SEED CTR 알고리즘 운영을 위한 구조체 지정  
(KISA\_SEED\_CTR\_init으로 초기화 필요)  
v\_out 사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼  
out\_offset 출력 버퍼의 시작 오프셋  
outLen 출력 버퍼에 저장된 데이터의 길이

반환값 :

- 1 : 패딩 성공
- 0 : 패딩 실패

## 테스트 페이지

### 국산 암호 [SEED-CTR] 테스트 페이지

<p>&lt;암호화 예제&gt;</p> <p>키(KEY): <input type="text" value="88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89"/></p> <p>카운터(CTR): <input type="text" value="00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE"/></p> <p>평문: <input type="text" value="00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F"/></p> <p><input type="button" value="▼ 암호화"/></p> <p>암호문: <input type="text" value="83,72,13,DF,61,DD,CB,5B,50,AC,EB,54,5B,86,43,ED"/></p>	<p>&lt;복호화 예제&gt;</p> <p>키(KEY): <input type="text" value="88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89"/></p> <p>카운터(CTR): <input type="text" value="00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE"/></p> <p>암호문: <input type="text"/></p> <p><input type="button" value="▼ 복호화"/></p> <p>평문: <input type="text"/></p>
---	---

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex : 00,01,0A,0B)

<키(KEY)> : 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89  
 <초기카운터> : 00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE  
 <평문> : 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F  
 <암호문> : 83,72,13,DF,61,DD,CB,5B,50,AC,EB,54,5B,86,43,ED

## 5.2. JSP

소스 활용 등 배포되는 소스코드를 이용하여 암호화/복호화를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, 운영체제 환경 및 Tomcat 설정에 대한 설명은 생략하기로 한다.

### 5.2.1 소스코드 추가

암호화/복호화 소스코드가 작성된 파일을 사용하고자 하는 파일에 포함 시킨다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="EUC-KR"%>
<%@ include file="KISA_SEED_ECB.jsp" %>
<!--
public byte[] getBytes(String data) {
    String[] str = data.split(",");
    byte[] result = new byte[str.length];
    for(int i=0; i<result.length; i++) {
        result[i] = getHex(str[i]);
    }
    return result;
}

public String getString(byte[] data) {
    String result = "";
    for(int i=0; i<data.length; i++) {
        result = result + toHex(data[i]);
        if(i<data.length-1)
            result = result + ",";
    }
    return result;
}
```

## 5.2.2 소스코드 설명

### o SEED-ECB

#### ☞ 함수 설명

```
public static void SEED_ECB_Encrypt(byte[] pbszUserKey, byte[] pbData, int offset, int length)
```

SEED-ECB 알고리즘 암호화

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbData	사용자 입력 평문
offset	사용자 입력 길이 시작 오프셋
length	사용자 입력 길이

#### 반환값 :

- 없음

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

```
public static void SEED_ECB_Decrypt(byte[] pbszUserKey, byte[] pbData, int offset, int length)
```

SEED-ECB 알고리즘 복호화

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbData	사용자 입력 평문
offset	사용자 입력 길이 시작 오프셋
length	사용자 입력 길이

#### 반환값 :

- 없음

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

🔗 테스트 페이지

## 국산 암호 [SEED-ECB] 테스트 페이지

<암호화 예제>

키(KEY):

평문:

▼암호화

암호문:

<복호화 예제>

키(KEY):

암호문:

▼복호화

평문:

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex : 00,01,0A,0B)

<키(KEY)> : 2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C

<평문> : 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F

<암호문> : F7,A5,AB,AA,86,9B,E1,1E,C1,D0,3B,BA,92,76,A1,64,AD,6C,74,B7,08,D1,CA,7E,B1,AA,FF,31,96,34,C6,02

## o SEED-CBC

### 함수 설명

```
public static int SEED_CBC_Encrypt(byte[] pbszUserKey, byte[] pbszIV, byte[] message, int message_offset, int message_length)
```

SEED-CBC 알고리즘 암호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
message_length	사용자 입력 길이

#### 반환값 :

- 암호화가 진행된 길이(byte 단위)

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 한다.

```
public static int SEED_CBC_Decrypt(byte[] pbszUserKey, byte[] pbszIV, byte[] pbszCipherText, int nCipherTextLen, byte[] nPlainTextLen)
```

SEED-CBC 알고리즘 복호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 암호문
message_offset	사용자 입력 길이 시작 오프셋
nPlainTextLen	사용자 입력 길이

#### 반환값 :

- 복호화가 진행된 길이(byte 단위)

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 한다.



```
public static int SEED_CBC_init( KISA_SEED_INFO pInfo, KISA_ENC_DEC enc, byte[] pbszUserKey, byte[] pbszIV)
```

SEED-CBC 알고리즘 초기화 함수

**매개변수 :**

pInfo            SEED CBC 알고리즘 운영을 위한 구조체 지정  
                  (미리 메모리가 할당되어 있어야 함)  
enc              알고리즘 암호화 및 복호화 모드 지정  
                  (암호화 : KISA\_ENC\_DEC.KISA\_ENCRYPT /  
                  복호화 : KISA\_ENC\_DEC.KISA\_DECRYPT)  
pbszUserKey    사용자가 지정하는 입력 키 (16 bytes)  
pbszIV          사용자가 지정하는 초기화 벡터 (16 bytes)

**반환값 :**

- 0 : KISA\_SEED\_INFO 구조체나 user\_key, iv가 널 포인터인 경우
- 1 : 초기화 성공

```
public static int SEED_CBC_Process( KISA_SEED_INFO pInfo, int[] in, int inLen, int[] out, int[] outLen )
```

SEED-CBC 알고리즘 다중 블록 암호화 함수

**매개변수 :**

pInfo            SEED CBC 알고리즘 운영을 위한 구조체 지정  
                  (KISA\_SEED\_CBC\_init으로 초기화 필요)  
in                사용자 입력 평문/암호문  
inLen            사용자 입력의 길이 지정 (char 단위)  
out              사용자 입력에 대한 암호문/평문 출력 버퍼  
outLen           출력 버퍼에 저장된 데이터의 길이 (byte 단위)

**반환값 :**

- 0 :            inLen의 값이 0보다 작은 경우,  
                  KISA\_SEED\_INFO 구조체나 in, out에 널 포인터가 할당되었을 경우
- 1 :            구동 성공

**참 고 :**

1. 출력이 되는 버퍼의 크기는 사용자 입력의 길이보다 크거나 같게 미리 할당해야 한다.
2. outLen은 실제로 출력버퍼 out에 저장된 결과 값의 길이를 함수 내부에서 지정 한다.

```
public static int SEED_CBC_Close(KISA_SEED_INFO pInfo, int[] out, int out_offset, int[] outLen)
```

SEED-CBC 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

#### 매개변수 :

pInfo SEED CBC 알고리즘 운영을 위한 구조체 지정  
(KISA\_SEED\_CBC\_init으로 초기화 필요)  
out 사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼  
outLen 출력 버퍼에 저장된 데이터의 길이 (char 단위)

#### 반환값 :

- 0 : out에 널 포인터가 할당되었을 경우  
복호화시 out값이 적절하지 않을 경우(패딩)
- 1 : 성공

#### 참 고 :

1. 출력버퍼 out은 SEED 알고리즘의 한 블록(16 bytes)이상으로 메모리가 할당되어 있어야 한다.

## 테스트 페이지

### 국산 암호 [SEED-CBC] 테스트 페이지

<암호화 예제>		<복호화 예제>	
키(KEY):	88, E3, 4F, 8F, 08, 17, 79, F1, E9, F3, 94, 37, 0A, D4, 05, 89	키(KEY):	
초기값(IV):	26, 8D, 66, A7, 35, A8, 1A, 81, 6F, BA, D9, FA, 36, 16, 25, 01	초기값(IV):	
평문:	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F	암호문:	
	<input type="button" value="▼암호화"/>		<input type="button" value="▼복호화"/>
암호문:	75, DD, A4, B0, 65, FF, 86, 42, 7D, 44, 8C, 54, 03, D3, 5A, 07, D3, 5A, AB, 86, 7C, 8B, F2, 55, 7D, 82, 38, 8E, A7, C0, D0, F1	평문:	

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex : 00,01,0A,0B)

<키(KEY)>:	88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89
<초기값(IV)>:	26,8D,66,A7,35,A8,1A,81,6F,BA,D9,FA,36,16,25,01
<평문>:	00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F
<암호문>:	75,DD,A4,B0,65,FF,86,42,7D,44,8C,54,03,D3,5A,07,D3,5A,AB,86,7C,8B,F2,55,7D,82,38,8E,A7,C0,D0,F1

## o SEED-CTR

### 함수 설명

```
public static int SEED_CTR_Encrypt(byte[] pbszUserKey, byte[] pbszCTR, byte[] message,
int message_offset, int nInputTextLen)
```

SEED-CTR 알고리즘 암호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 평문
message_offset	사용자 입력 길이 시작 오프셋
nInputTextLen	사용자 입력 길이

#### 반환값 :

- 암호화가 진행된 길이(byte 단위)

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼의 크기와 동일( 패딩 처리를 하지 않는다. )

```
public static int SEED_CTR_Decrypt( byte[] pbszUserKey, byte[] pbszCTR, byte[] message,
int message_offset, int nInputTextLen)
```

SEED-CTR 알고리즘 복호화 함수

#### 매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
message	사용자 입력 암호문
message_offset	사용자 입력 길이 시작 오프셋
nInputTextLen	사용자 입력 길이

#### 반환값 :

- 복호화가 진행된 길이(byte 단위)

#### 참 고 :

1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. pbszIV 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼 크기와 동일하다.

```
public static int SEED_CTR_init(KISA_SEED_INFO pInfo, KISA_ENC_DEC enc, byte[]
pszUserKey, byte[] pbszCounter)
```

SEED-CTR 알고리즘 초기화 함수

**매개변수 :**

pInfo                SEED CBC 알고리즘 운영을 위한 구조체 지정  
(미리 메모리가 할당되어 있어야 함)  
enc                    알고리즘 암호화 및 복호화 모드 지정  
(암호화 : KISA\_ENC\_DEC.KISA\_ENCRYPT /  
복호화 : KISA\_ENC\_DEC.KISA\_DECRYPT)  
pbszUserKey        사용자가 지정하는 입력 키 (16 bytes)  
pbszCounter        사용자가 지정하는 초기 카운터(16 bytes)

**반환값 :**

- 0 : KISA\_SEED\_INFO 구조체나 user\_key, iv가 널 포인터인 경우
- 1 : 초기화 성공

```
public static int SEED_CTR_Process(KISA_SEED_INFO pInfo, int[] in, int inLen, int[] out,
int[] outLen)
```

SEED-CTR 알고리즘 다중 블록 암호화 함수

**매개변수 :**

pInfo                SEED CTR 알고리즘 운영을 위한 구조체 지정  
(KISA\_SEED\_init으로 초기화 필요)  
in                    사용자 입력 평문/암호문  
inLen                사용자 입력의 길이 지정 (char 단위)  
out                   사용자 입력에 대한 암호문/평문 출력 버퍼  
outLen               출력 버퍼에 저장된 데이터의 길이 (char 단위)

**반환값 :**

- 0 : inLen의 값이 0보다 작은 경우,  
KISA\_SEED\_INFO 구조체나 in, out에 널 포인터가 할당되었을 경우
- 1 : 구동 성공

**참 고 :**

1. 출력이 되는 버퍼의 크기는 사용자 입력의 길이보다 크거나 같게 미리 할당 해야 함
2. outLen은 실제로 출력버퍼 out에 저장된 결과값의 길이를 함수 내부에서 지정함

```
public function SEED_CTR_Close(KISA_SEED_INFO pInfo, int[] out, int out_offset, int[] outLen)
```

SEED-CTR 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

매개변수 :

pInfo SEED CTR 알고리즘 운영을 위한 구조체 지정  
(KISA\_SEED\_CTR\_init으로 초기화 필요)  
out 사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼  
out\_offset 출력 버퍼의 시작 오프셋  
outLen 출력 버퍼에 저장된 데이터의 길이

반환값 :

- 1 : 패딩 성공
- 0 : 패딩 실패

## 테스트 페이지

### 국산 암호 [SEED-CTR] 테스트 페이지

<p>&lt;암호화 예제&gt;</p> <p>키(KEY): 88, E3, 4F, 8F, 08, 17, 79, F1, E9, F3, 94, 37, 0A, D4, 05, 89</p> <p>초기카운터(ctr): 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, FE</p> <p>평문: 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F</p> <p>▼암호화</p> <p>암호문: 83, 72, 13, DF, 61, DD, CB, 5B, 50, AC, EB, 54, 5B, 86, 43, ED</p>	<p>&lt;복호화 예제&gt;</p> <p>키(KEY):</p> <p>초기카운터(ctr):</p> <p>암호문:</p> <p>▼복호화</p> <p>평문:</p>
--	--

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex : 00,01,0A,0B)

<키(KEY)> : 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89  
<초기카운터(ctr)> : 00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE  
<평문> : 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F  
<암호문> : 83,72,13,DF,61,DD,CB,5B,50,AC,EB,54,5B,86,43,ED

## 5.3. PHP

소스 활용 등 배포되는 소스코드를 이용하여 암호화/복호화를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, Apache 서버 설정에 대한 설명은 생략하기로 한다.

### 5.3.1 소스코드 추가

암호화/복호화 소스코드가 작성된 파일을 사용하고자 하는 파일에 포함 시킨다.

```
1  <?
2  require_once ('KISA_SEED_ECB.php');
3
4  $g_bszUser_key = $_POST['KEY'];
5
6  if($g_bszUser_key == null)
7  {
8      $g_bszUser_key = "2b,7e,1f,16,28,ae,d2,a6,ab,f7,15,88,09,cf,4f,3c";
9  }
10
11 function encrypt($bszUser_key, $str) {
12
13     $planBytes = split(",",$str);
14     $keyBytes = split(",",$bszUser_key);
15
16     for($i = 0; $i < 16; $i++)
17     {
18         $keyBytes[$i] = hexdec($keyBytes[$i]);
19     }
20     for ($i = 0; $i < count($planBytes); $i++) {
21         $planBytes[$i] = hexdec($planBytes[$i]);
22     }
23
24     if (count($planBytes) == 0) {
25         return $str;
26     }
27 }
```

### 5.3.2 소스코드 설명

#### o SEED-ECB

##### 함수 설명

<code>static function SEED_ECB_Encrypt(&amp;\$pbszUserKey, &amp;\$message, \$message_offset, \$message_length)</code>
---

SEED-ECB 알고리즘 암호화 함수

##### 매개변수 :

<code>\$pbszUserKey</code>	사용자가 지정하는 입력 키 (16 bytes)
<code>\$message</code>	사용자 입력 평문
<code>\$message_offset</code>	사용자 입력 길이 시작 오프셋
<code>\$message_length</code>	사용자 입력 길이

##### 반환값 :

- 사용자 입력에 대한 암호문 출력 버퍼

##### 참 고 :

1. `pbszUserKey` 의 크기는 반드시 16 bytes 여야 한다.

<code>static function SEED_ECB_Decrypt(&amp;\$pbszUserKey, &amp;\$message, \$message_offset, \$message_length)</code>
---

SEED-ECB 알고리즘 복호화 함수

##### 매개변수 :

<code>\$pbszUserKey</code>	사용자가 지정하는 입력 키 (16 bytes)
<code>\$message</code>	사용자 입력 암호문
<code>\$message_offset</code>	사용자 입력 길이 시작 오프셋
<code>\$message_length</code>	사용자 입력 길이

##### 반환값 :

- 사용자 입력에 대한 평문 출력 버퍼

##### 참 고 :

1. `pbszUserKey` 의 크기는 반드시 16 bytes 여야 한다.

## 국산 암호 [SEED-ECB] 테스트 페이지

	<p>&lt;암호화 예제&gt;</p> <p>키(KEY): 2b, 7e, 15, 16, 28, ae, d2, a6, ab, f7, 15, 88, 09, cf, 4f, 3c</p> <p>평문: 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F</p> <p>암호문: <input type="button" value="▼ 암호화"/></p> <p>F7, A5, AB, AA, 86, 9B, E1, 1E, C1, D0, 3B, BA, 92, 76, A1, 64, AD, 6C, 74, B7, 08, D1, CA, 7E, B1, AA, FF, 31, 96, 34, C6, 02</p>	<p>&lt;복호화 예제&gt;</p> <p>키(KEY): 2b, 7e, 15, 16, 28, ae, d2, a6, ab, f7, 15, 88, 09, cf, 4f, 3c</p> <p>암호문: </p> <p>평문: <input type="button" value="▼ 복호화"/></p>
--	--	--

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex : 00,01,0A,0B)

<키(KEY)> : 2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C

<평문> : 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F

<암호문> : F7,A5,AB,AA,86,9B,E1,1E,C1,D0,3B,BA,92,76,A1,64,AD,6C,74,B7,08,D1,CA,7E,B1,AA,FF,31,96,34,C6,02



## o SEED-CBC

### 함수 설명

```
static function SEED_CBC_Encrypt(&$pbszUserKey, &$pbszIV, &$message, $message_offset, $message_length )
```

SEED-CBC 알고리즘 암호화 함수

#### 매개변수 :

\$pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
\$pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
\$message	사용자 입력 평문
\$message_offset	사용자 입력 길이 시작 오프셋
\$message_length	사용자 입력 길이

#### 반환값 :

- 사용자 입력에 대한 암호문 출력 버퍼

#### 참 고 :

1. \$pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. \$pbszIV 의 크기는 반드시 16 bytes 여야 한다.

```
static function SEED_CBC_Decrypt(&$pbszUserKey, &$pbszIV, &$message, $message_offset, $message_length )
```

SEED-CBC 알고리즘 복호화 함수

#### 매개변수 :

\$pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
\$pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)
\$message	사용자 입력 암호문
\$message_offset	사용자 입력 길이 시작 오프셋
\$message_length	사용자 입력 길이

#### 반환값 :

- 사용자 입력에 대한 평문 출력 버퍼

#### 참 고 :

1. \$pbszUserKey 의 크기는 반드시 16 bytes 여야 하다.
2. \$pbszIV 의 크기는 반드시 16 bytes 여야 하다.

```
static function SEED_CBC_init(&$pInfo, $enc, &$pbszUserKey, &$pbszIV )
```

SEED-CBC 알고리즘 초기화 함수

**매개변수 :**

\$pInfo	SEED CBC 알고리즘 운영을 위한 구조체 지정 (미리 메모리가 할당되어 있어야 함)
\$enc	알고리즘 암호화 및 복호화 모드 지정 (암호화 : KISA_ENCRYPT / 복호화 : KISA_DECRYPT)
\$pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
\$pbszIV	사용자가 지정하는 초기화 벡터 (16 bytes)

**반환값 :**

- 1 : 초기화 성공
- 0 : 초기화 실패

```
static function SEED_CBC_Process( &$pInfo, &$in, $inLen, &$out, &$outLen )
```

SEED-CBC 알고리즘 다중 블록 암호화 함수

**매개변수 :**

\$pInfo	SEED CBC 알고리즘 운영을 위한 구조체 지정 (KISA_SEED_CBC_init으로 초기화 필요)
\$in	사용자 입력 평문/암호문
\$inLen	사용자 입력의 길이 지정
\$out	사용자 입력에 대한 암호문/평문 출력 버퍼
\$outLen	출력 버퍼에 저장된 데이터의 길이

**반환값 :**

- 1 : 구동 성공
- 0 : 구동 실패

static function SEED\_CBC\_Close( &\$pInfo, &\$out, \$out\_offset, &\$outLen )

SEED-CBC 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

매개변수 :

\$pInfo	SEED CBC 알고리즘 운영을 위한 구조체 지정 (KISA_SEED_CBC_init으로 초기화 필요)
\$out	사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼
\$out_offset	출력 버퍼의 시작 오프셋
\$outLen	출력 버퍼에 저장된 데이터의 길이

반환값 :

- 1 : 패딩 성공
- 0 : 패딩 실패

## 테스트 페이지

### 국산 암호 [SEED-CBC] 테스트 페이지

<p>&lt;암호화 예제&gt;</p> <p>키(KEY): 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89</p> <p>초기값(IV): 26,8D,66,A7,35,A8,1A,81,6F,BA,D9,FA,36,16,25,01</p> <p>평문: 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F</p> <p>▼ 암호화</p> <p>암호문: 75,DD,A4,B0,65,FF,86,42,7D,44,8C,54,03,D3,5A,07,D3,5A,AB,86,7C,8B,F2,55,7D,82,38,8E,A7,C0,D0,F1</p>	<p>&lt;복호화 예제&gt;</p> <p>키(KEY): 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89</p> <p>초기값(IV): 26,8D,66,A7,35,A8,1A,81,6F,BA,D9,FA,36,16,25,01</p> <p>암호문:</p> <p>▼ 복호화</p> <p>평문:</p>
--	--

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex: 00,01,0A,0B)

<키(KEY)> : 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89  
 <초기값(IV)> : 26,8D,66,A7,35,A8,1A,81,6F,BA,D9,FA,36,16,25,01  
 <평문> : 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F  
 <암호문> : 75,DD,A4,B0,65,FF,86,42,7D,44,8C,54,03,D3,5A,07,D3,5A,AB,86,7C,8B,F2,55,7D,82,38,8E,A7,C0,D0,F1

o SEED-CTR

static	function	SEED_CTR_Encrypt(&\$pbszUserKey,	&\$pbszCTR,	&\$message,
		\$message_offset, \$message_length)		

SEED-CTR 알고리즘 암호화 함수

**매개변수 :**

\$pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
\$pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
\$message	사용자 입력 평문
\$message_offset	사용자 입력 길이 시작 오프셋
\$message_length	사용자 입력 길이

**반환값 :**

- 사용자 입력에 대한 암호문 출력 버퍼

**참 고 :**

1. \$pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. \$pbszIV 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼의 크기와 동일( 패딩 처리를 하지 않는다. )

static	function	SEED_CTR_Decrypt(&\$pbszUserKey,	&\$pbszCTR,	&\$message,
		\$message_offset, \$message_length )		

SEED-CTR 알고리즘 복호화 함수

**매개변수 :**

\$pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
\$pbszCTR	사용자가 지정하는 초기화 벡터 (16 bytes)
\$message	사용자 입력 암호문
\$message_offset	사용자 입력 길이 시작 오프셋
\$message_length	사용자 입력 길이

**반환값 :**

- 사용자 입력에 대한 평문 출력 버퍼

**참 고 :**

1. \$pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.
2. \$pbszIV 의 크기는 반드시 16 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼 크기와 동일하다.

static function SEED\_CTR\_init(&\$pInfo, \$enc, &\$pbszUserKey, &\$pbszCTR )

SEED-CTR 알고리즘 초기화 함수

**매개변수 :**

\$pInfo	SEED CBC 알고리즘 운영을 위한 구조체 지정 (미리 메모리가 할당되어 있어야 함)
\$enc	알고리즘 암호화 및 복호화 모드 지정 (암호화 : KISA_ENCRYPT / 복호화 : KISA_DECRYPT)
\$pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
\$pbszCTR	사용자가 지정하는 초기 카운터(16 bytes)

**반환값 :**

- 1 : 초기화 성공
- 0 : 초기화 실패

static function SEED\_CTR\_Process( &\$pInfo, &\$in, \$inLen, &\$out, &\$outLen )

SEED-CTR 알고리즘 다중 블록 암호화 함수

**매개변수 :**

\$pInfo	SEED CTR 알고리즘 운영을 위한 구조체 지정 (KISA_SEED_init으로 초기화 필요)
\$in	사용자 입력 평문/암호문
\$inLen	사용자 입력의 길이 지정 (char 단위)
\$out	사용자 입력에 대한 암호문/평문 출력 버퍼
\$outLen	출력 버퍼에 저장된 데이터의 길이 (char 단위)

**반환값 :**

- 1 : 구동 성공
- 0 : 구동 실패

**참 고 :**

1. 출력이 되는 버퍼의 크기는 사용자 입력의 길이보다 크거나 같게 미리 할당해야 함
2. outLen은 실제로 출력버퍼 out에 저장된 결과값의 길이를 함수 내부에서 지정함

```
static function SEED_CTR_Close(&$pInfo, &$out, $out_offset, &$outLen)
```

SEED-CTR 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수

**매개변수 :**

pInfo	SEED CTR 알고리즘 운영을 위한 구조체 지정 (KISA_SEED_CTR_init으로 초기화 필요)
out	사용자 입력에 대한 최종 출력 블록이 저장되는 버퍼
out_offset	출력 버퍼의 시작 오프셋
outLen	출력 버퍼에 저장된 데이터의 길이

**반환값 :**

- 1 : 패딩 성공
- 0 : 패딩 실패

## ☞ 테스트 페이지

### 국산 암호 [SEED-CTR] 테스트 페이지

<p>&lt;암호화 예제&gt;</p> <p>키(KEY): 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89</p> <p>카운터(CTR): 00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE</p> <p>평문: 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F</p> <p>▼ 암호화</p> <p>암호문: 83,72,13,df,61,dd,cb,5b,50,ac,eb,54,5b,86,43,ed</p>	<p>&lt;복호화 예제&gt;</p> <p>키(KEY): 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89</p> <p>카운터(CTR): 00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE</p> <p>암호문: 83,72,13,df,61,dd,cb,5b,50,ac,eb,54,5b,86,43,ed</p> <p>▼ 복호화</p> <p>평문: 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F</p>
--	--

※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex : 00,01,0A,0B)

```
<키(KEY)>: 88,E3,4F,8F,08,17,79,F1,E9,F3,94,37,0A,D4,05,89
<초기카운터>: 00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE
<평문>: 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F
<암호문>: 83,72,13,DF,61,DD,CB,5B,50,AC,EB,54,5B,86,43,ED
```

## [부록]

### 참조구현값

본 SEED 소스코드 매뉴얼에서는 ECB, CBC, CTR 운영모드 표준의 구현적합성 실험을 위한 참조구현값(Test Vectors)이 제공된다.

참조구현값을 생성하기 위한 평문 데이터와 키, 초기값, 초기 카운트는 아래 표와 같다. 데이터 1(512비트)에 대한 참조구현값은 암호화 연산에서 단계별 입력 블록(I), 출력 블록(O)의 구체적인 중간값(Intermediate Value)이 추가로 제공되며, 데이터 2 (1,280비트)에 대한 참조구현값은 평문 데이터를 암호화한 암호문 데이터만 제공된다.

#### < 데이터 1 - 512비트 >

키(Key)		88 E3 4F 8F 08 17 79 F1 E9 F3 94 37 0A D4 05 89
초기값(IV)		26 8D 66 A7 35 A8 1A 81 6F BA D9 FA 36 16 25 01
초기 카운터(ctr)		00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE
평문(P)	블록1	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
	블록2	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
	블록3	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
	블록4	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3

#### < 데이터 2 - 1,280비트 >

키(Key)		ED 24 01 AD 22 FA 25 59 91 BA FD B0 1F EF D6 97
초기값(IV)		93 EB 14 9F 92 C9 90 5B AE 5C D3 4D A0 6C 3C 8E
초기 카운터(ctr)		FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
평문(P)	블록1	B4 0D 70 03 D9 B6 90 4B 35 62 27 50 C9 1A 24 57
	블록2	5B B9 A6 32 36 4A A2 6E 3A C0 CF 3A 9C 9D 0D CB
	블록3	38 13 33 2C 97 15 E7 BB 9F 1C 34 A6 6B 8A 8F 93
	블록4	77 DC A1 A8 71 EF 3F 72 10 92 65 56 DE 48 C0 DC
	블록5	47 31 6C 66 B4 36 92 D5 92 9C 2A 35 F3 E5 63 8D
	블록6	6E B1 32 C1 7A B6 E1 53 3B F3 50 3C B4 B2 17 13
	블록7	8F 8A 8A B8 F8 92 29 CC 22 EE BB 14 42 76 EE 86
	블록8	E5 71 B4 FA 5F 95 15 93 DC F8 91 BD 67 E5 51 1A
	블록9	8D 06 00 FF A3 73 26 A7 4E 08 CA 60 25 2C F7 6A
	블록10	7A 00 FD D6 C4 0C BB 0C B4 03 12 6E EF E2 7B 85

## 1 ECB 모드 참조구현값

### 1.1 데이터 1 - 암호화

블록1	평문(P1) 입력(I1) 출력(O1) 암호문(C1)	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F 0F 4E 7F C7 8C 48 D5 AD 95 10 BA D8 98 7B A5 22 0F 4E 7F C7 8C 48 D5 AD 95 10 BA D8 98 7B A5 22
블록2	평문(P2) 입력(I2) 출력(O2) 암호문(C2)	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96 8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96 39 86 9D 94 74 48 58 38 24 99 67 68 D1 31 F0 A5 39 86 9D 94 74 48 58 38 24 99 67 68 D1 31 F0 A5
블록3	평문(P3) 입력(I3) 출력(O3) 암호문(C3)	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93 39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93 D6 8C 46 19 B7 C3 45 A7 80 CB 8E 16 77 0B 25 9A D6 8C 46 19 B7 C3 45 A7 80 CB 8E 16 77 0B 25 9A
블록4	평문(P4) 입력(I4) 출력(O4) 암호문(C4)	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3 D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3 36 8C B3 C5 B5 B1 2F FD 78 08 6F D0 5F 39 FC DC 36 8C B3 C5 B5 B1 2F FD 78 08 6F D0 5F 39 FC DC

### 1.2 데이터 1 - 복호화

블록1	암호문(C1) 입력(I1) 출력(O1) 평문(P1)	0F 4E 7F C7 8C 48 D5 AD 95 10 BA D8 98 7B A5 22 0F 4E 7F C7 8C 48 D5 AD 95 10 BA D8 98 7B A5 22 D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
블록2	암호문(C2) 입력(I2) 출력(O2) 평문(P2)	39 86 9D 94 74 48 58 38 24 99 67 68 D1 31 F0 A5 39 86 9D 94 74 48 58 38 24 99 67 68 D1 31 F0 A5 8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96 8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
블록3	암호문(C3) 입력(I3) 출력(O3) 평문(P3)	D6 8C 46 19 B7 C3 45 A7 80 CB 8E 16 77 0B 25 9A D6 8C 46 19 B7 C3 45 A7 80 CB 8E 16 77 0B 25 9A 39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93 39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
블록4	암호문(C4) 입력(I4) 출력(O4) 평문(P4)	36 8C B3 C5 B5 B1 2F FD 78 08 6F D0 5F 39 FC DC 36 8C B3 C5 B5 B1 2F FD 78 08 6F D0 5F 39 FC DC D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3 D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3



### 1.3 데이터 2 - 암호화

블록1	암호문(C1)	C0 92 AC 0B AA 9A 98 B0 6F 53 E0 37 0A EB 2B A2
블록2	암호문(C2)	68 41 00 CC 59 42 B0 25 D8 C3 0E 67 DF 16 D5 FA
블록3	암호문(C3)	7D AE 6E 42 70 EE 4D F6 9B 4F B5 27 74 5A 74 E5
블록4	암호문(C4)	D9 E3 4B 2A F6 3B 64 5B 4F 5A A8 96 68 FD 61 24
블록5	암호문(C5)	BA 47 DA 33 22 86 72 CC C1 FD F1 F3 20 23 9D 35
블록6	암호문(C6)	9E C9 27 C0 FF 1F 51 F8 98 7C FF 13 0D 7C EA 9E
블록7	암호문(C7)	79 8C 01 DC C3 80 F1 34 50 79 9C 48 3D A1 1A 24
블록8	암호문(C8)	34 75 A6 41 19 3F 1D 72 13 1B CC 7C CF E2 20 F6
블록9	암호문(C9)	EE B2 79 86 8D A1 60 12 AB 68 69 1D 0D AB D3 B7
블록10	암호문(C10)	60 FB E9 5F 88 54 AD 8A 76 7F 40 25 8D C0 4F 1D

### 1.4 데이터 2 - 복호화

블록1	암호문(C1)	B4 0D 70 03 D9 B6 90 4B 35 62 27 50 C9 1A 24 57
블록2	암호문(C2)	5B B9 A6 32 36 4A A2 6E 3A C0 CF 3A 9C 9D 0D CB
블록3	암호문(C3)	38 13 33 2C 97 15 E7 BB 9F 1C 34 A6 6B 8A 8F 93
블록4	암호문(C4)	77 DC A1 A8 71 EF 3F 72 10 92 65 56 DE 48 C0 DC
블록5	암호문(C5)	47 31 6C 66 B4 36 92 D5 92 9C 2A 35 F3 E5 63 8D
블록6	암호문(C6)	6E B1 32 C1 7A B6 E1 53 3B F3 50 3C B4 B2 17 13
블록7	암호문(C7)	8F 8A 8A B8 F8 92 29 CC 22 EE BB 14 42 76 EE 86
블록8	암호문(C8)	E5 71 B4 FA 5F 95 15 93 DC F8 91 BD 67 E5 51 1A
블록9	암호문(C9)	8D 06 00 FF A3 73 26 A7 4E 08 CA 60 25 2C F7 6A
블록10	암호문(C10)	7A 00 FD D6 C4 0C BB 0C B4 03 12 6E EF E2 7B 85

## 2 CBC 모드 참조구현값

### 2.1 데이터 1 - 암호화

블록1	평문(P1) 입력(I1) 출력(O1) 암호문(C1)	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F F1 E0 6B BF 07 D6 DF E3 DE E4 B2 39 53 BA 29 0E A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27 A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27
블록2	평문(P2) 입력(I2) 출력(O2) 암호문(C2)	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96 2F D2 0A 52 4A 2B D7 02 DC 47 E3 A6 CD 8B 44 B1 E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34 E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34
블록3	평문(P3) 입력(I3) 출력(O3) 암호문(C3)	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93 D1 FA 26 2B B5 AE 7D 6B 6F 6E 89 1C 57 C0 10 A7 CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79 CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79
블록4	평문(P4) 입력(I4) 출력(O4) 암호문(C4)	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3 1A 7F 56 E6 89 9A 6D B9 59 25 56 D5 64 DC 4A CA C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06 C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06

### 2.2 데이터 1 - 복호화

블록1	암호문(C1) 입력(I1) 출력(O1) 평문(P1)	A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27 A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27 F1 E0 6B BF 07 D6 DF E3 DE E4 B2 39 53 BA 29 0E D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
블록2	암호문(C2) 입력(I2) 출력(O2) 평문(P2)	E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34 E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34 2F D2 0A 52 4A 2B D7 02 DC 47 E3 A6 CD 8B 44 B1 8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
블록3	암호문(C3) 입력(I3) 출력(O3) 평문(P3)	CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79 CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79 D1 FA 26 2B B5 AE 7D 6B 6F 6E 89 1C 57 C0 10 A7 39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
블록4	암호문(C4) 입력(I4) 출력(O4) 평문(P4)	C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06 C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06 1A 7F 56 E6 89 9A 6D B9 59 25 56 D5 64 DC 4A CA D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3

### 2.3 데이터 2 - 암호화

블록1	암호문(C1)	F0 72 C5 B1 A0 58 8C 10 5A F8 30 1A DC D9 1D D0
블록2	암호문(C2)	67 F6 82 21 55 30 4B F3 AA D7 5C EB 44 34 1C 25
블록3	암호문(C3)	D8 68 F1 11 A8 2F 6F 4D 51 41 6F 21 5C 88 10 06
블록4	암호문(C4)	EB 2B E8 53 25 C1 A2 19 60 CF C8 97 4C 31 E1 9E
블록5	암호문(C5)	6B 4E 11 93 1B 8C 28 A4 5B A9 BE 66 53 DE 5E 11
블록6	암호문(C6)	92 18 91 0B C9 A2 05 0B 40 A5 5B 0D 91 7B 26 9F
블록7	암호문(C7)	2A 0F 1D 3F E0 41 E6 83 AE F4 DE D2 14 71 A8 05
블록8	암호문(C8)	06 34 00 0C 2E 2B 87 E4 C9 A2 2C 05 24 7A A4 0D
블록9	암호문(C0)	14 2D 91 5D 84 4F 1E 3E 6E 6C DD E3 DF 50 DD 88
블록10	암호문(C10)	94 7B D6 46 68 EB CD 35 86 5B D2 3A 37 02 FA 04

### 2.4 데이터 2 - 복호화

블록1	암호문(C1)	B4 0D 70 03 D9 B6 90 4B 35 62 27 50 C9 1A 24 57
블록2	암호문(C2)	5B B9 A6 32 36 4A A2 6E 3A C0 CF 3A 9C 9D 0D CB
블록3	암호문(C3)	38 13 33 2C 97 15 E7 BB 9F 1C 34 A6 6B 8A 8F 93
블록4	암호문(C4)	77 DC A1 A8 71 EF 3F 72 10 92 65 56 DE 48 C0 DC
블록5	암호문(C5)	47 31 6C 66 B4 36 92 D5 92 9C 2A 35 F3 E5 63 8D
블록6	암호문(C6)	6E B1 32 C1 7A B6 E1 53 3B F3 50 3C B4 B2 17 13
블록7	암호문(C7)	8F 8A 8A B8 F8 92 29 CC 22 EE BB 14 42 76 EE 86
블록8	암호문(C8)	E5 71 B4 FA 5F 95 15 93 DC F8 91 BD 67 E5 51 1A
블록9	암호문(C0)	8D 06 00 FF A3 73 26 A7 4E 08 CA 60 25 2C F7 6A
블록10	암호문(C10)	7A 00 FD D6 C4 0C BB 0C B4 03 12 6E EF E2 7B 85

### 3 CTR 모드 참조구현값

#### 3.1 데이터 1 - 암호화

블록1	평문(P1) 입력(I1) 출력(O1) 암호문(C1)	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE 83 73 11 DC 65 D8 CD 5C 58 A5 E1 5F 57 8B 4D E2 54 1E 1C C4 57 A6 08 3E E9 FB 8A 9C 32 27 41 ED
블록2	평문(P2) 입력(I2) 출력(O2) 암호문(C2)	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 19 7B CC 56 B6 DF F4 9E 13 2C AC FD 28 75 55 D3 94 3A 2C ED 25 5A 9C 30 F8 D1 3E 10 32 8A F5 45
블록3	평문(P3) 입력(I3) 출력(O3) 암호문(C3)	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 15 1F FD 7E 39 93 2C 79 5B 0F 8D 05 FE 27 30 C8 2C 52 DD 82 6B E4 F1 85 16 E7 3D F9 1F CC 1B 5B
블록4	평문(P4) 입력(I4) 출력(O4) 암호문(C4)	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01 0F A3 40 18 62 4C 97 1B 96 77 E2 51 A7 31 4C B6 DB 0D 00 F7 25 24 51 08 23 7C 6B 13 50 E5 F5 05

#### 3.2 데이터 1 - 복호화

블록1	암호문(C1) 입력(I1) 출력(O1) 평문(P1)	54 1E 1C C4 57 A6 08 3E E9 FB 8A 9C 32 27 41 ED 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE 83 73 11 DC 65 D8 CD 5C 58 A5 E1 5F 57 8B 4D E2 D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
블록2	암호문(C2) 입력(I2) 출력(O2) 평문(P2)	94 3A 2C ED 25 5A 9C 30 F8 D1 3E 10 32 8A F5 45 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 19 7B CC 56 B6 DF F4 9E 13 2C AC FD 28 75 55 D3 8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
블록3	암호문(C3) 입력(I3) 출력(O3) 평문(P3)	2C 52 DD 82 6B E4 F1 85 16 E7 3D F9 1F CC 1B 5B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 15 1F FD 7E 39 93 2C 79 5B 0F 8D 05 FE 27 30 C8 39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
블록4	암호문(C4) 입력(I4) 출력(O4) 평문(P4)	DB 0D 00 F7 25 24 51 08 23 7C 6B 13 50 E5 F5 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01 0F A3 40 18 62 4C 97 1B 96 77 E2 51 A7 31 4C B6 D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3

### 3.3 데이터 2 - 암호화

블록1	암호문(C1)	87 F9 6C 34 C4 97 A1 79 5D 4E 38 81 02 53 37 26
블록2	암호문(C2)	64 3E 8C 7C 3E 69 F7 6C 73 23 AF 9D 0C 76 BA 76
블록3	암호문(C3)	5F 8A 64 E7 C2 11 B6 8E C4 CC 3C 6B 7D DD 63 8E
블록4	암호문(C4)	1D 76 F5 49 83 9E 13 A8 C2 94 33 7B F7 09 EE D2
블록5	암호문(C5)	F3 B8 BE F2 2A E5 5B 66 41 1B FD 05 3F 66 28 C7
블록6	암호문(C6)	37 41 2C 99 34 C4 52 D6 6A 0F AA CB EA 42 20 D9
블록7	암호문(C7)	7C E8 69 C2 E4 58 FA 51 88 32 BE C9 D2 59 34 0C
블록8	암호문(C8)	E7 3E 19 EA AF DE E7 E8 1D 2D DB 37 32 9C AD 8E
블록9	암호문(C9)	21 BD 45 E3 70 73 73 A6 32 C9 E4 68 3E 9A DD F8
블록10	암호문(C10)	B3 62 57 A2 73 06 ED 09 3A DA FE E0 8A 50 CE 98

### 3.4 데이터 2 - 복호화

블록1	암호문(C1)	B4 0D 70 03 D9 B6 90 4B 35 62 27 50 C9 1A 24 57
블록2	암호문(C2)	5B B9 A6 32 36 4A A2 6E 3A C0 CF 3A 9C 9D 0D CB
블록3	암호문(C3)	38 13 33 2C 97 15 E7 BB 9F 1C 34 A6 6B 8A 8F 93
블록4	암호문(C4)	77 DC A1 A8 71 EF 3F 72 10 92 65 56 DE 48 C0 DC
블록5	암호문(C5)	47 31 6C 66 B4 36 92 D5 92 9C 2A 35 F3 E5 63 8D
블록6	암호문(C6)	6E B1 32 C1 7A B6 E1 53 3B F3 50 3C B4 B2 17 13
블록7	암호문(C7)	8F 8A 8A B8 F8 92 29 CC 22 EE BB 14 42 76 EE 86
블록8	암호문(C8)	E5 71 B4 FA 5F 95 15 93 DC F8 91 BD 67 E5 51 1A
블록9	암호문(C9)	8D 06 00 FF A3 73 26 A7 4E 08 CA 60 25 2C F7 6A
블록10	암호문(C10)	7A 00 FD D6 C4 0C BB 0C B4 03 12 6E EF E2 7B 85