HIGHT 블록암호 알고리즘에 대한 소스코드 활용 매뉴얼

2018.11.

KISA 한국인터넷진흥원
Korea Internet & Security Agency

제 · 개 정 이 력

순번	제 · 개정일	제 · 개정 내역	담당자
1	2013.12	“HIGHT 블록암호 알고리즘에 대한 소스코드 활용” 매뉴얼	김기문
2	2018.11	HIGHT 블록암호 운영모드(CMAC) 추가	김준섭
3			
4			

< 목 차 >

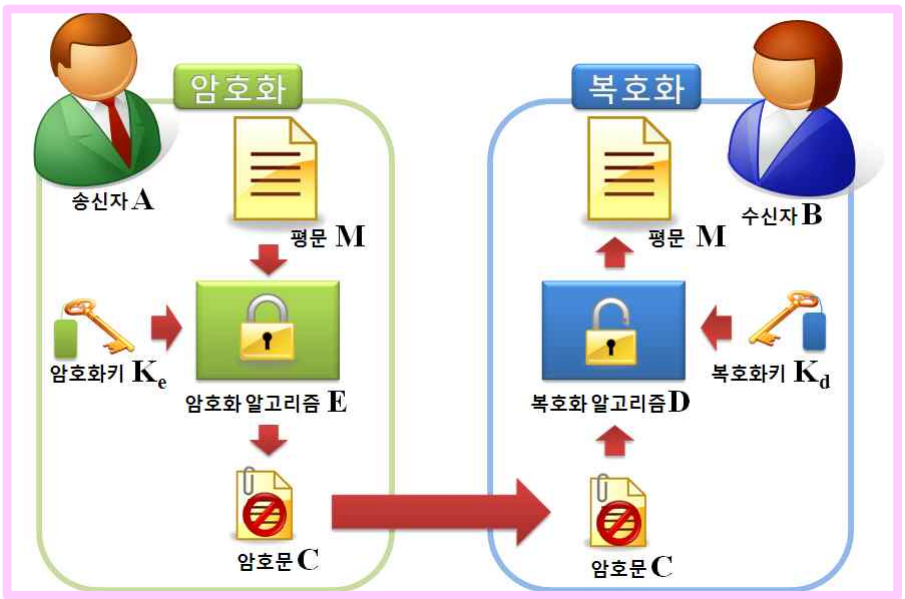
1. 개요

HIGHT 암호 알고리즘 HIGHT(HIGh security and light weigHT)는 RFID, USN 등과 같이 저전력·경량화를 요구하는 컴퓨팅 환경에서 기밀성을 제공하기 위해 2005년 KISA, ETRI 부설연구소 및 고려대가 공동으로 개발한 64비트 블록암호 알고리즘이다.

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

2. 블록암호 알고리즘

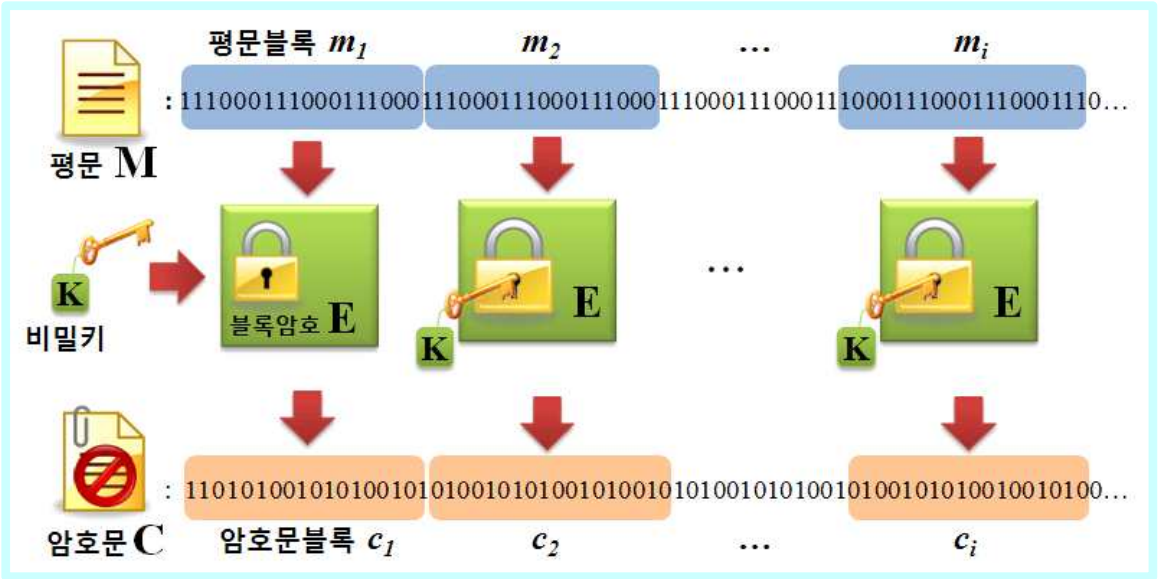
암호(Cryptography)란 메시지를 해독 불가능한 형태로 변환하거나 또는 암호화된 메시지를 해독 가능한 형태로 변환하는 기술을 말한다. 이때, 해독 가능한 형태의 메시지를 평문(Plaintext)이라고 하고, 해독 불가능한 형태의 메시지를 암호문(Ciphertext)이라 하며, 평문을 암호문으로 변환하는 과정을 암호화(Encryption), 암호문을 평문으로 변환하는 과정을 복호화(Decryption)라고 한다. 정보를 숨기고, 숨긴 정보를 볼 수 있기 위해서는 권한이 있는 사람만이 암호화 및 복호화를 할 수 있어야 한다. 그래서 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.



< 암·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 >

HIGHT는 64비트의 암·복호화키를 이용하여 임의의 길이를 갖는 입력 메시지를 64비트의 블록단위로 처리하는 64비트 블록암호 알고리즘이다. 따라서 임의의 길이를 가지는 평문 메시지를 64비트씩 블록단위로 나누어 암호화하여 암호문을 생성한다.

3. 고려사항

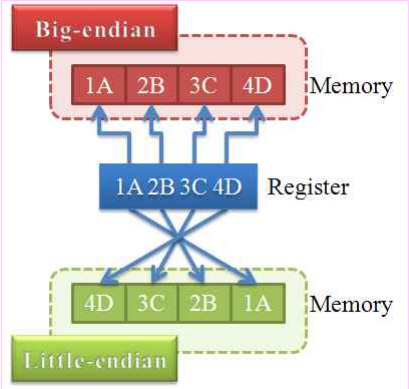
블록암호 HIGHT를 실제로 사용하기 위해서는 구현 환경에서 사용되는 엔디안(endianness)과 암호화된 데이터가 저장되는 데이터 형식(Data Type)을 고려해야 한다. 또한, 입력 블록을 블록암호에 적용하는 방법인 운영모드(Mode of operation)와 마지막 블록의 크기를 맞추기 위한 패딩(Padding)을 함께 구현해 주어야 한다.

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, HIGHT 소스코드도 기본적으로 리틀 엔디안을 사용하여 설명한다.

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

< 엔디안에 따른 표현 >



엔디안은 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 || ₩
    __OS2__ || sun386 || __TURBOC__ || vax || vms || VMS || __VMS
#define LITTLE_ENDIAN
#else
#define BIG_ENDIAN
#endif
```

자바 가상 머신은 기본적으로 빅 엔디안을 사용하기 때문에, Java 소스코드에서는 기본 엔디안으로 빅 엔디안이 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 HEX 형태로 처리해 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. 하지만 이번 매뉴얼에서는 KISA에서 개발하여 배포하는 ECB, CBC, CTR에 대해서만 설명을 하도록 하겠다.

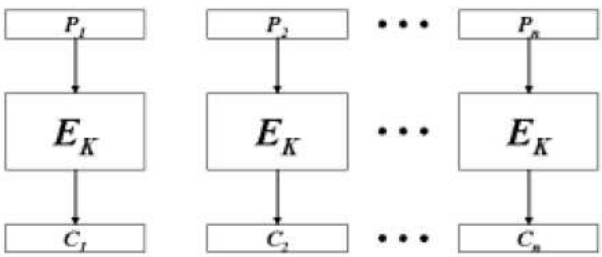
3.3.1 ECB 운영모드

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

ECB 모드의 암호화 과정은 평문 블록(Pi)을 입력 블록(Ii)으로 설정하고, 이를 암호화한 출력 블록(Oi)을 암호문 블록(Ci)으로 설정한다.

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

< ECB 암호화 의사코드 >

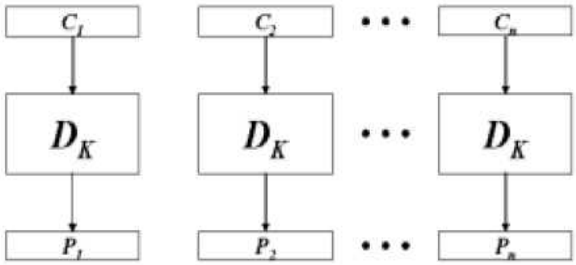


< ECB 암호화 >

ECB 모드의 복호화 과정은 평문 블록(Ci)을 입력 블록(Ii)으로 설정하고, 이를 암호화한 출력 블록(Oi)을 암호문 블록(Pi)으로 설정한다.

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

< ECB 복호화 의사코드 >



< ECB 복호화 >

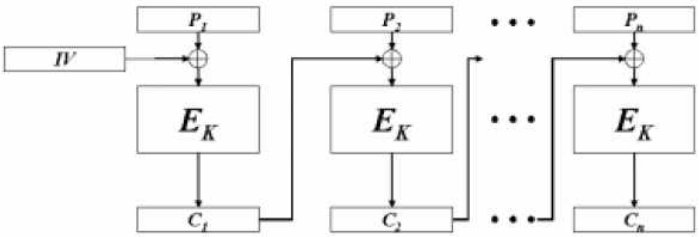
3.3.2 CBC 운영모드

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

CBC 모드의 암호화 과정은 현 단계에서 평문 블록(Pi)과 전 단계의 암호문 블록(Ci-1)을 배타적 논리합 연산한 결과를 현 단계의 입력 블록(Ii)으로 설정하고, 이를 암호화한 출력 블록(Oi)을 현 단계의 암호문 블록(Ci)으로 설정한다.

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

< CBC 암호화 의사코드 >

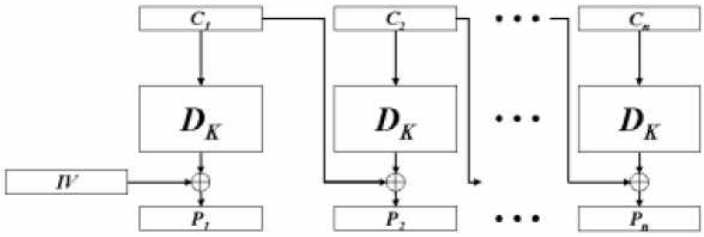


< CBC 암호화 >

CBC 모드의 복호화 과정은 현 단계의 암호문 블록(Ci)을 입력 블록(Ii)으로 설정하고, 이를 복호화한 출력 블록(Oi)을 전 단계의 입력 블록(Ii-1)인 암호문 블록(Ci-1)과 배타적 논리합 연산한 결과를 현 단계의 평문 블록(Pi)으로 한다.

o 입력 : $C=\{C_1,\cdots,C_n\}$, K, IV
o 출력 : $P=\{P_1,\cdots,P_n\}$
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>

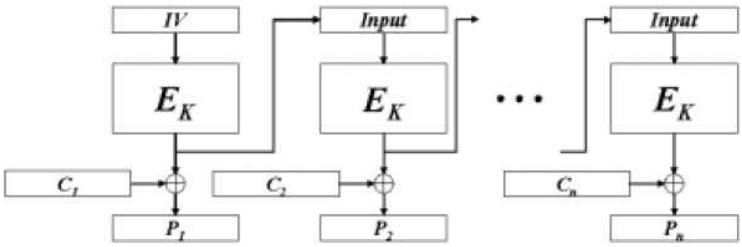
3.3.3 CTR 운영모드

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(Ii)으로 하여 암호화한 출력 블록(Oi)을 평문 블록(Pi)과 배타적 논리합 연산을 수행함으로 암호문 블록(Ci)을 생성한다.

o 입력 : $P=\{P_1,\cdots,P_n\}$, K, ctr
o 출력 : $C=\{C_1,\cdots,C_n\}$
o 처리과정
for i=1 to n
$I_i = ctr+(i-1) \bmod 2^{64}$
$O_i = E_K(I_i)$
$C_i = P_i \oplus O_i$

< CTR 암호화 의사코드 >

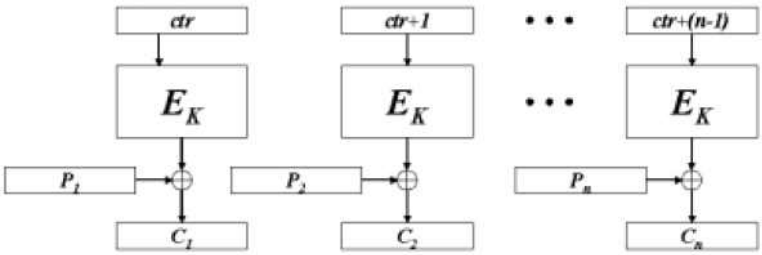


< CTR 암호화 >

CTR 모드의 복호화 과정은 카운터(ctr+(i-1))를 현 단계의 입력 블록(Ii)으로 하여 암호화한 출력 블록(Oi)을 암호문 블록(Ci)과 배타적 논리합 연산을 수행함으로 평문 블록(Pi)을 생성한다.

o 입력 : $C=\{C_1,\cdots,C_n\}$, K, ctr
o 출력 : $P=\{P_1,\cdots,P_n\}$
o 처리과정
for i=1 to n
$I_i = ctr+(i-1) \bmod 2^{64}$
$O_i = E_K(I_i)$
$P_i = C_i \oplus O_i$

< CTR 복호화 의사코드 >

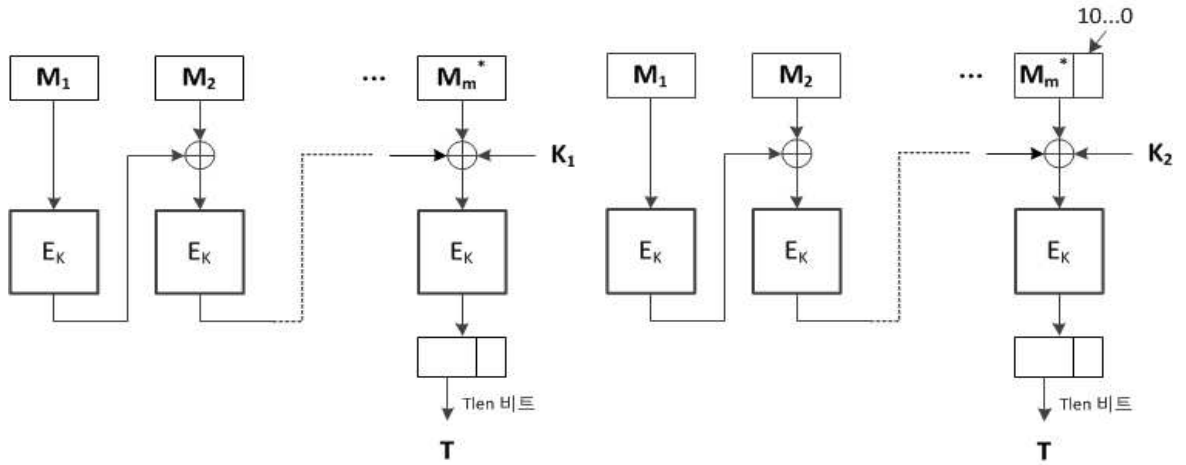


< CTR 복호화 >

3.3.4 CMAC 운영모드

CMAC은 CBC 모드로부터 파생된 메시지 인증 방식 CBC-MAC을 안전성 측면에서 개선시킨 운영모드이다.

CMAC의 인증값 생성 과정은 먼저 메시지(M)를 b 비트 단위로 분할하며, 분할된 각 블록을 차례대로 M_1, M_2, …, M_{m-1}, M_m^* 으로 표기한다. 여기에서 블록 M_m^*의 길이는 b 비트 이하이다. 그리고 (m-1)개의 메시지 블록 열 M_1, M_2, … , M_{m-1}에서 현재 단계의 메시지 블록(M_i)과 직전 단계 블록암호 암호화 함수 출력 블록의 XOR 결과를 암호화 함수의 입력 블록으로 설정하여 현재 단계 출력 블록을 얻는 구성을 반복한다. 마지막 메시지 블록(M_m^*)을 처리할 때 해당 메시지 블록의 길이가 b 비트인지 여부에 따라 비밀 키 K로부터 생성한 보조 비밀 키(K1 또는 K2)와 직전 단계 암호화 함수 출력 블록, 그리고 메시지 블록(M_m^*)의 XOR 결과를 블록암호 암호화 함수 입력 블록으로 설정한다. 이로부터 얻은 블록암호 암호화 함수 출력 블록의 상위 Tlen비트를 인증값으로 설정한다.



Len(M_m^*) = b일 때 Len(M_m^*) ≠ b일 때

< 보조 비밀 키를 사용한 CMAC 인증값 생성 >

---

1 입력 : 메시지 $M = M_1 \| M_2 \| \cdots \| M_m^*$ ($Len(M_i) = b$ $(1 \le i \le (m-1))$, $Len(M_m^*) \le b$),
메시지 길이 Mlen,
비밀 키 K,
인증값 길이 Tlen

※ 보조 비밀키 생성요소 R_b ($R_{128}=0^{120}\|10000111$, $R_{64}=0^{59}\|11011$)

---

2 처리 과정

(1) 보조 비밀 키 K1, K2 생성:

$L \leftarrow E_K(0^b)$;

if $MSB_1(L) = 0$ then

K1 ← L ≪ 1;
else
K1 ← (L ≪ 1) ⊕ R_b;
endif
if MSB_1(K1) = 0 then
K2 ← K1 ≪ 1;
else
K2 ← (K1 ≪ 1) ⊕ R_b;
endif
(2) 인증값 계산:
if Mlen = 0 then
m ← 1;
else
m ← ⌈ Mlen/b⌉ ;
endif

if Len(M_m^*) = b then
M_m ← K1 ⊕ M_m^*;
else
M_m ← K2 ⊕ (M_m^* ‖ 10^j), where j = b − Len(M_m^*) − 1;
endif
Y ← 0^b;
for i from 1 to m do
X ← Y ⊕ M_i;
Y ← E_K(X);
endfor
T ← MSB_{Tlen}(Y);

3 출력 : 인증값 T

< CMAC 인증값 생성 의사코드 >

수신한 메시지 M과 인증값 T가 유효한지를 검증하기 위해서는 비밀 키 K를 사용해서 직접 인증값 T_1을 계산한 후 T_1 = T 여부를 확인한다.

3.4. 패딩방법

메시지를 HIGHT에 입력하기 위해 여러 개의 64비트 블록으로 나눌 때, 마지막 블록을 정확히 64비트 블록의 크기로 맞추는 것은 쉽지 않을 것이다. 예를 들어, 300비트의 메시지를 64비트의 블록으로 나눌 경우, 130 = 64 + 64 + 2 로 나뉘어 세 개의 블록을 구성하게 되는 데, 이때 마지막 블록이 2비트로 64비트를 만족하지 못한다. 이 경우 나머지 부족한 62비트를 채워주어야 HIGHT의 입력값으로 사용할

수 있다. 이렇게 부족한 부분을 채우는 방식을 패딩이라고 한다.

ECB, CBC 모드는 평문 블록을 암호화의 입력으로 사용하기 때문에, 평문 데이터의 크기가 64비트의 양의 정수배가 되어야 하기 때문에, 반드시 덧붙이기 방법이 적용되어야만 한다. 그러나, CFB-s, OFB, CTR 모드의 경우, 마지막 암호화 단계의 평문 블록이 64비트(CFB-s 모드의 경우 s비트)를 만족하지 못하고 m비트가 남아있을 때, 마지막 출력 블록(On) 64비트 중 상위 m비트 (MSBs(On))와 마지막 평문 블록 m비트와 배타적 논리합 연산함으로 아래와 같은 덧붙이기 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 64비트 양의 정수배가 아닐 때, 마지막 평문 블록이 64비트가 되도록 바이트 '00'을 덧붙인다.

```
예) 입력 블록(48비트) : 4F 52 49 54 48 4D
    패딩 블록(64비트) : 4F 52 49 54 48 4D 00 00

    입력 블록(64비트) : 53 45 45 44 41 4C 47 A8
    패딩 블록(64비트) : 53 45 45 44 41 4C 47 A8
```

3.4.2 패딩 방법 2

평문 데이터의 크기가 64비트의 양의 정수배가 아닐 경우, 마지막 평문 블록이 64비트가 되도록 평문 데이터의 끝에 비트 '80' 추가한 후 나머지에 모두 '0' 비트를 덧붙이고, 평문 블록의 크기가 64비트의 양의 정수배일 경우, 추가적인 128비트 '80…00' 블록을 추가한다.

```
예) 입력 블록(48비트)  : 4F 52 49 54 48 4D
    패딩 블록(64비트)  : 4F 52 49 54 48 4D 80 00

    입력 블록(64비트)  : 53 45 45 44 41 4C 47 A8
    패딩 블록(128비트) : 53 45 45 44 41 4C 47 A8 80 00 00 00 00 00 00 00
```

3.4.3 패딩 방법 3

평문 데이터의 크기가 64비트의 양의 정수배가 아닐 경우, 마지막 평문 블록이 64비트가 되도록 덧붙이기가 필요한 바이트 수 'xx…xx'를 덧붙이고, 평문 블록의 크기가 64비트의 양의 정수배일 경우, 추가적인 64비트 '08…08' 블록을 추가한다.

```
예) 입력 블록(48비트) : 4F 52 49 54 48 4D
    패딩 블록(64비트) : 4F 52 49 54 48 4D 02 02

    입력 블록(64비트)  : 53 45 45 44 41 4C 47 A8
    패딩 블록(128비트) : 53 45 45 44 41 4C 47 A8 08 08 08 08 08 08 08 08
```

3.5. 활용 방법

HIGHT 알고리즘은 8Byte의 비밀키, 8Byte의 평문을 입력으로 8Byte의 암호문을 출력하는 블록암호 알고리즘이다.

입력	출력	키 길이	라운드
64 bits (8 bytes)	64 bits (8 bytes)	128 bits (16 bytes)	32 라운드

소스 코드에는 2가지 방법의 함수가 포함되어 있다.

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

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

4. 응용 프로그램

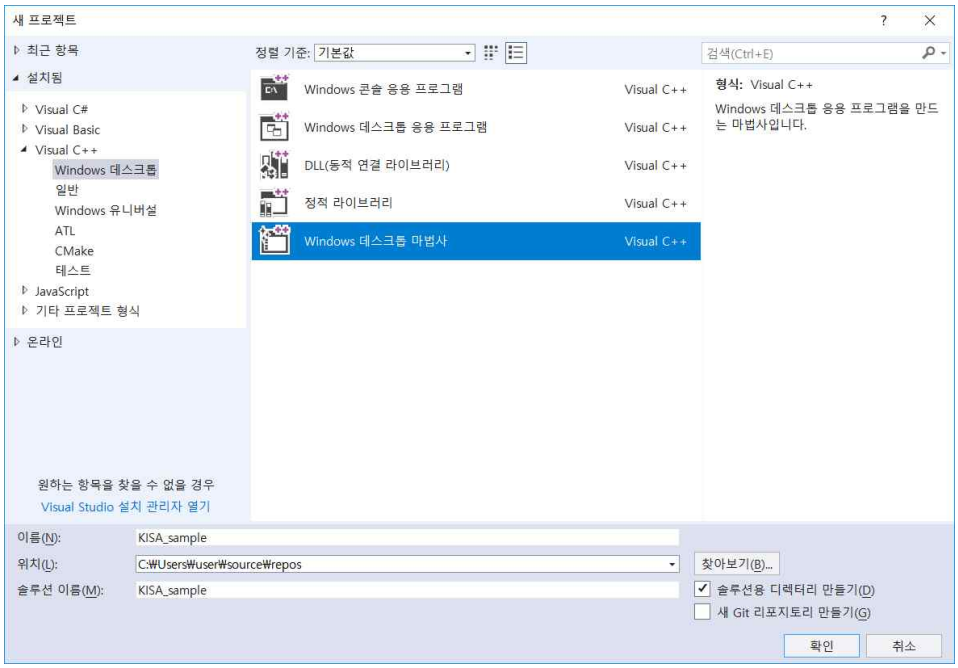
Windows, Linux 등 다양한 운영체제 환경에서 응용 프로그램을 개발과 함께 암호화를 적용할 수 있도록 국산암호 소스코드를 개발하였습니다. 기본적으로 많이 사용하는 C/C++ 및 Java를 기반으로 하며 이번 소스코드에 포함된 운영모드는 ECB, CBC, CTR, CMAC이 함께 제공됩니다.

4.1. C/C++

프로젝트 생성 및 소스 추가, 빌드 등 배포되는 소스코드를 이용하여 암호화/복호화를 실행하는 방법에 대해서는 Microsoft 社의 Visual studio 2017을 활용하여 설명하도록 한다.

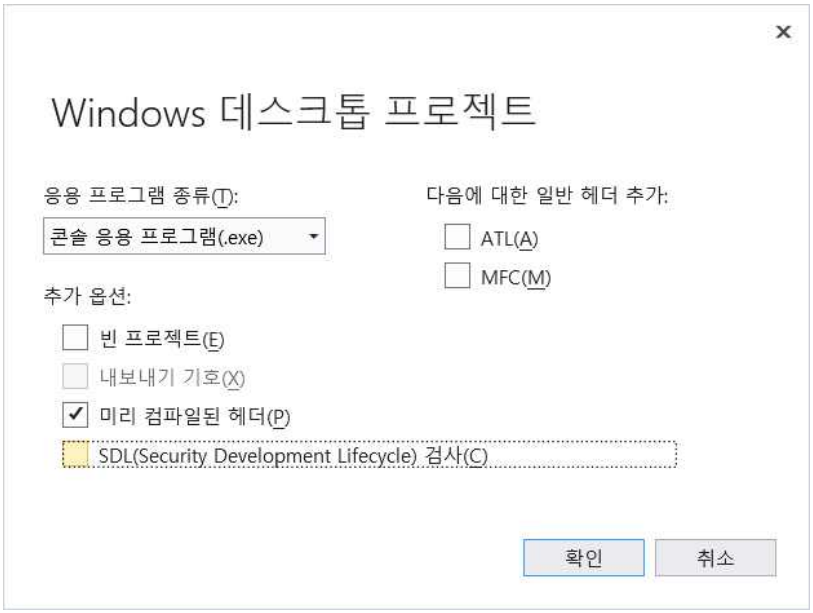
4.1.1 프로젝트 생성 및 빌드

Visual studio를 실행하여 “Windows 데스크톱 마법사”을 선택한다. 이름에는 프로젝트 명을 기입하고 위치에는 생성시키고자 하는 곳의 위치를 지정하여 준다.

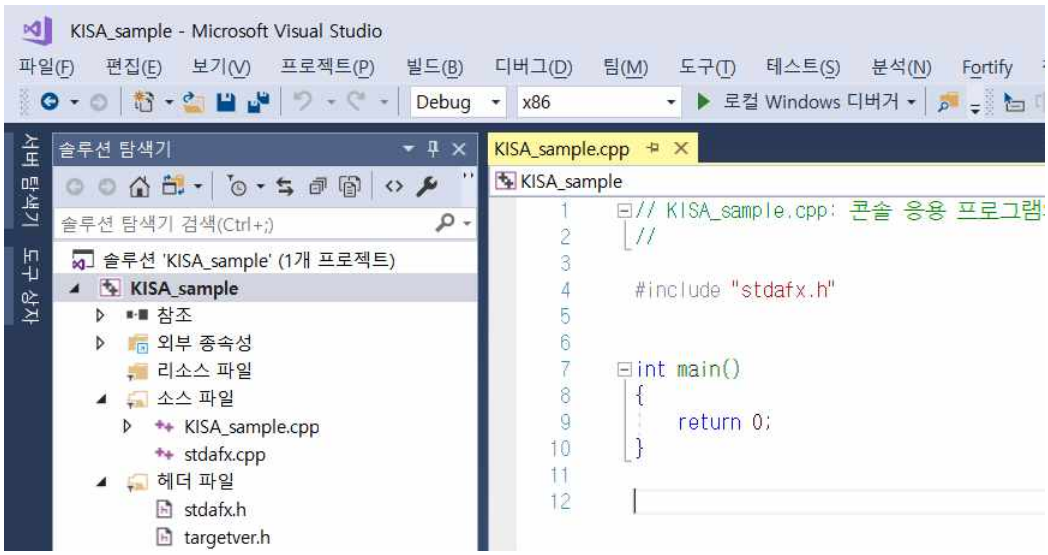


“콘솔 응용 프로그램”으로 선택한 후 마침 버튼을 누르면 콘솔 형태의 프로젝트가 생성된다. 미리 컴파일된 헤더를 체크해주면 int _tmain(int argc, TCHAR *argv[])을 자동으로 생성 시켜 준다.

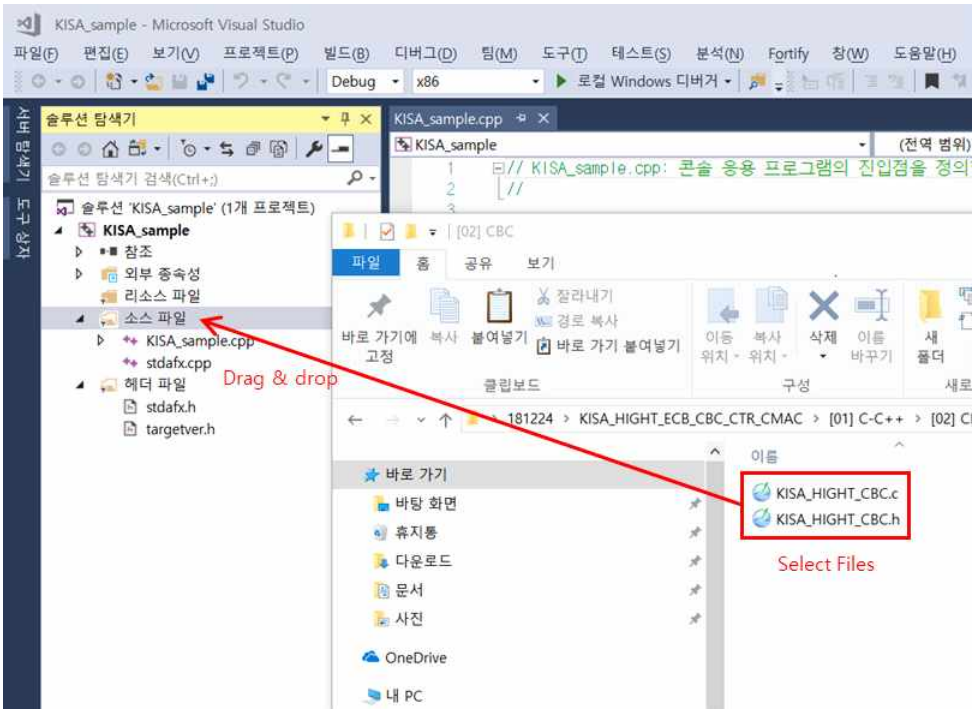
빈 프로젝트는 말그대로 아무것도 없는 것으로 직접 헤더 파일과 소스파일을 추가하여 전부 작성하는 것이다. 여기서는 미리 컴파일된 헤더를 사용하여 프로젝트를 구성하도록 한다.



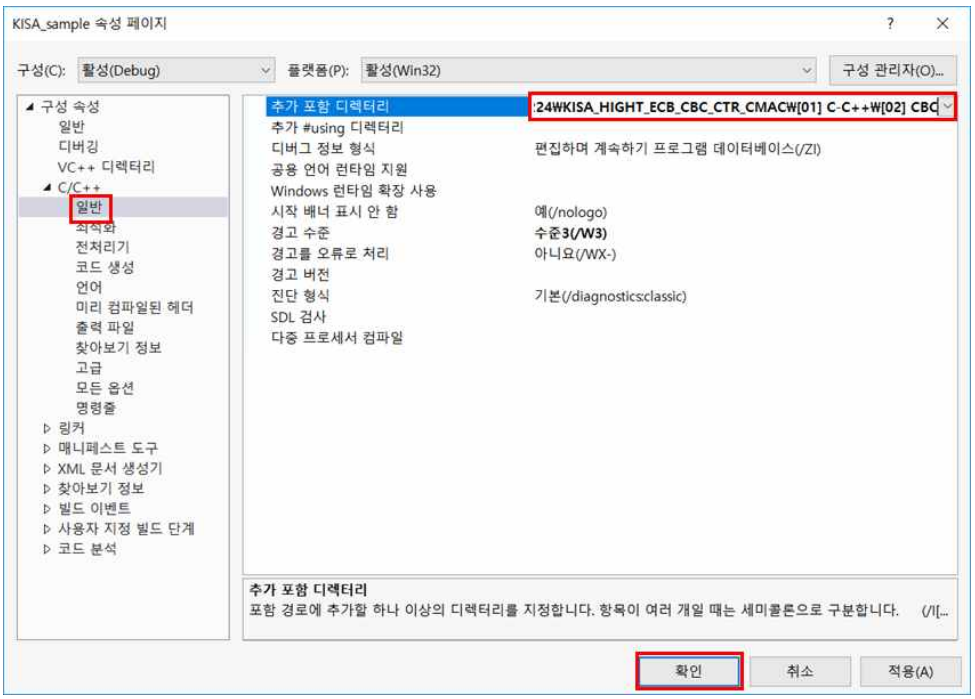
프로젝트 생성이 완료되면 “콘솔 응용 프로그램“을 작성할 수 있도록 기본적인 .h와 .cpp의 파일들을 생성된다. KISA_sample.cpp의 _tmain 함수 내에서 암호화/복호화 소스를 활용하도록 한다.



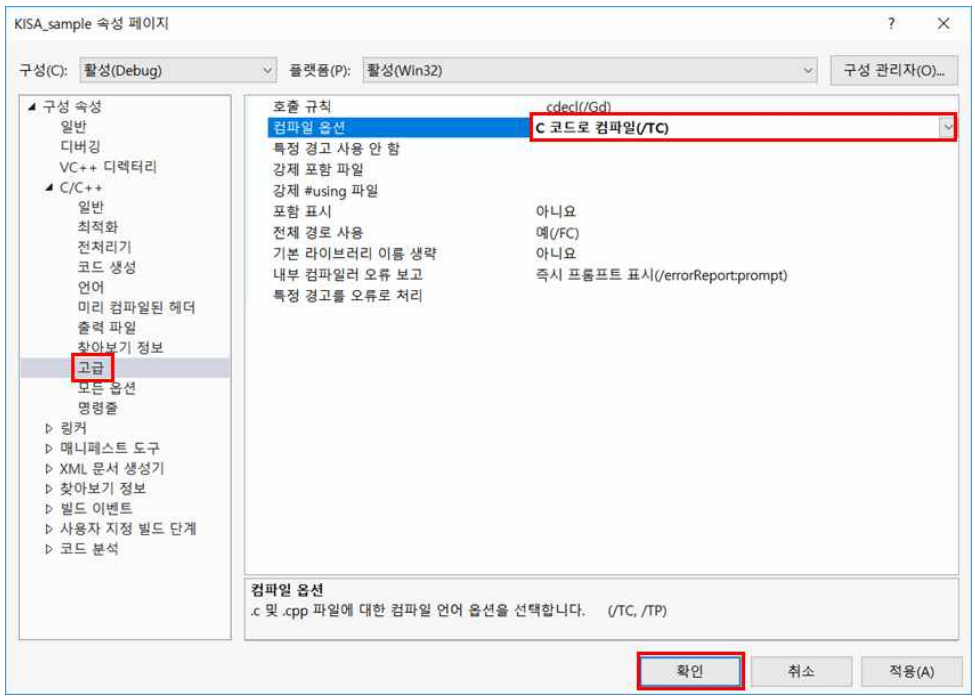
프로젝트 생성이 완료된 후, KISA에서 배포하는 소스코드 ”KISA_HIGHT_xxx.c“ 및 ”KISA_HIGHT_xxx.h“ 파일을 드래그 앤 드롭으로 소스 파일 폴더로 복사한다.



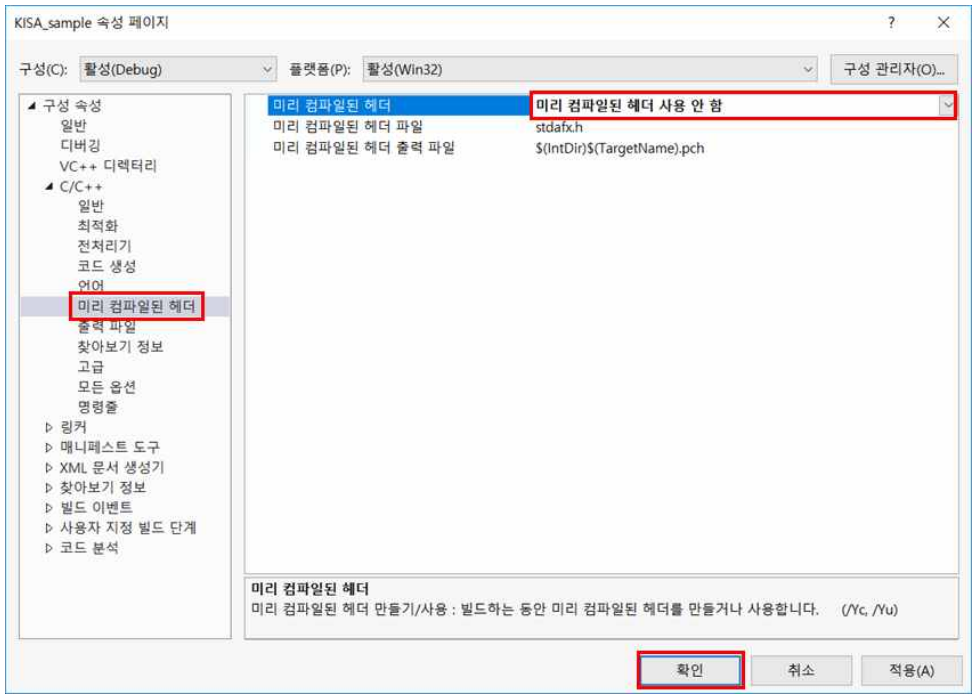
프로젝트 속성에서 C/C++의 추가 포함 디렉터리 항목에 포함 시킨 헤더와 소스 파일이 있는 폴더의 절대 경로 또는 상대 경로를 포함시켜야 한다. 여러 개의 폴더 또한 포함 시킬 수 있으며 구분자는 ";" 로 하여 연속하여 기입 하면 된다.



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 HIGHT-ECB

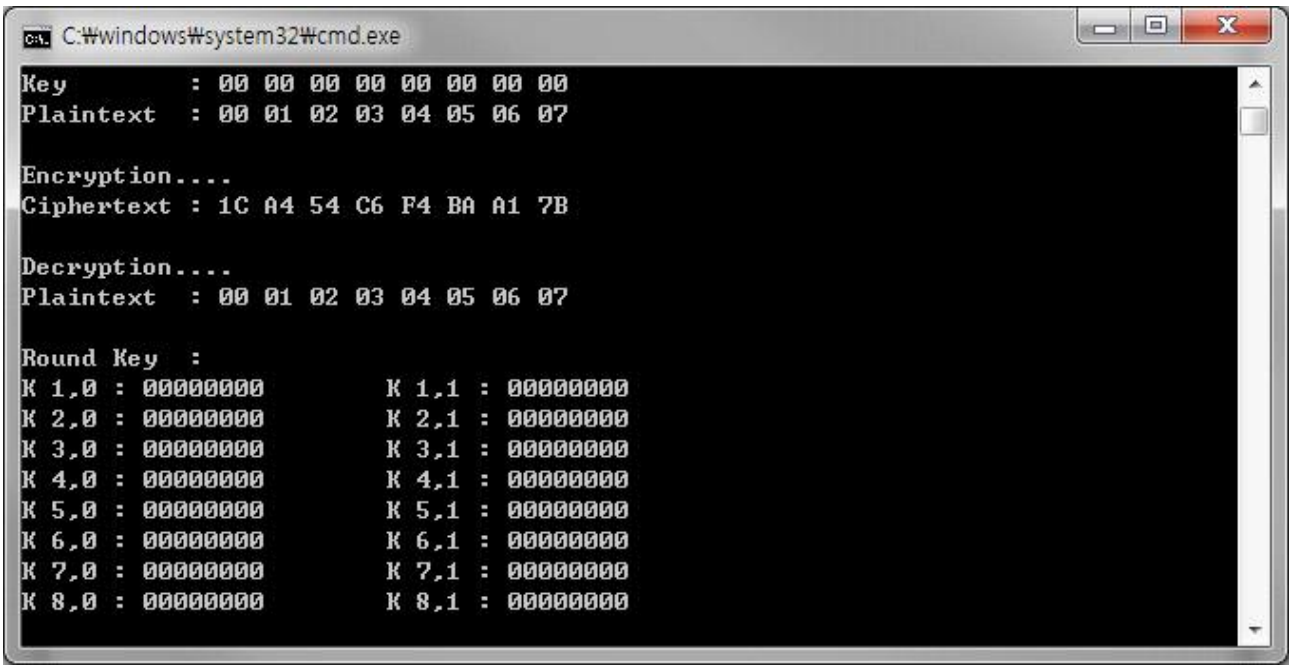
☞ **함수 설명**

void HIGHT_KeySched(BYTE *UserKey, DWORD UserKeyLen, BYTE *RoundKey);
HIGHT-ECB Round Key 생성 함수 **매개변수 :** UserKey　　사용자가 지정하는 입력 키 UserKeyLen　사용자가 지정하는 입력 키의 길이 RoundKey　　라운드키 생성 **반환값 :** - 없음

void HIGHT_Encrypt (BYTE *RoundKey, BYTE *Data);
HIGHT-ECB 알고리즘 암호화 함수 **매개변수 :** RoundKey　　HIGHT_KeySched에 의해 생성된 키 Data　　암호화할 데이터 (8 bytes) **반환값 :** - 없음 **참 고 :** 1. RoundKey 는 사전에 HIGHT_KeySched에 의해 생성되어 있어야 한다.

void HIGHT_Decrypt (BYTE *RoundKey, BYTE *Data);
HIGHT-ECB 알고리즘 복호화 함수 **매개변수 :** RoundKey　　HIGHT_KeySched에 의해 생성된 키 Data　　복호화할 데이터 (8 bytes) **반환값 :** - 없음 **참 고 :** 1. RoundKey 는 사전에 HIGHT_KeySched에 의해 생성되어 있어야 한다.

☞ 테스트 페이지


C:₩windows₩system32₩cmd.exe
Key : 00 00 00 00 00 00 00 00
Plaintext : 00 01 02 03 04 05 06 07
Encryption....
Ciphertext : 1C A4 54 C6 F4 BA A1 7B
Decryption....
Plaintext : 00 01 02 03 04 05 06 07
Round Key :
K 1,0 : 00000000 K 1,1 : 00000000
K 2,0 : 00000000 K 2,1 : 00000000
K 3,0 : 00000000 K 3,1 : 00000000
K 4,0 : 00000000 K 4,1 : 00000000
K 5,0 : 00000000 K 5,1 : 00000000
K 6,0 : 00000000 K 6,1 : 00000000
K 7,0 : 00000000 K 7,1 : 00000000
K 8,0 : 00000000 K 8,1 : 00000000

o HIGHT-CBC

☞ 함수 설명

int HIGHT_CBC_Encrypt(IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE *pbszPlainText, IN int nPlainTextLen, OUT BYTE *pbszCipherText)
HIGHT-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 8바이트 블럭으로 처리함으로 pbszCipherText는 평문보다 8바이트 커야 한다. (평문이 8바이트 블럭 시 패딩 데이타가 8바이트가 들어간다.)

int HIGHT_CBC_Decrypt(IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE *pbszCipherText, IN int nCipherTextLen, OUT BYTE *pbszPlainText)
HIGHT-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 HIGHT_CBC_init(OUT KISA_HIGHT_INFO *pInfo, IN KISA_ENC_DEC enc, IN BYTE *pUserKey, IN BYTE *pbIV)

HIGHT-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(8 BYTE)

반환값 :

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

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

HIGHT-CBC 다중 블럭 암호화/복호화 함수

매개변수 :

pInfo	HIGHT_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 HIGHT_CBC_Close(OUT KISA_SEED_INFO *pInfo, IN DWORD *out, IN int *outLen)

HIGHT-CBC 운영모드 종료 및 패딩 처리(PKCS7)

매개변수 :

pInfo	HIGHT_CBC_Process 를 거친 KISA_HIGHT_INFO 구조체
out	평문/암호문 출력 버퍼
outLen	출력 버퍼에 저장된 데이터 길이(BYTE 단위의 평문길이)

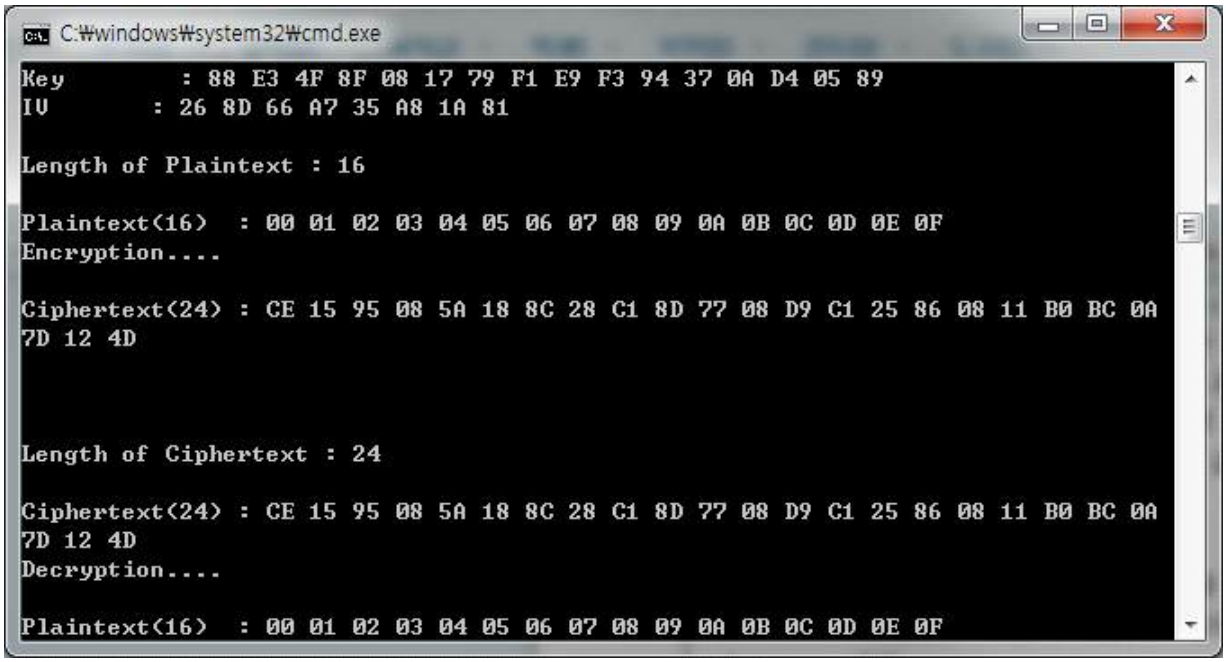
반환값 :

- 0 : inLen의 값이 0보다 작은 경우,
  KISA_SEED_INFO 구조체나 out에 널 포인터가 할당되었을 경우
- 1 : 성공

참 고 :

패딩 로직때문에 8바이트 블럭으로 처리함으로 복호화 시 출력 버퍼는 평문보다 8바이트 커야 한다.(평문이 8바이트 블럭 시 패딩 데이타가 8바이트가 들어간다.)

☞ **테스트 페이지**



o HIGHT-CTR

☞ **함수 설명**

int HIGHT_CTR_Encrypt(IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE *pbszPlainText, IN int nPlainTextLen, OUT BYTE *pbszCipherText)
HIGHT-CTR 알고리즘 암호화 함수(처리하고자 하는 데이터가 적을 경우에 사용) **매개변수 :** pbszUserKey 사용자가 지정하는 입력 키(16 BYTE) pszbIV 사용자가 지정하는 초기화 벡터(8 BYTE) pbszPlainText 사용자 입력 평문 nPlainTextLen 평문 길이(BYTE 단위의 평문길이) pbszCipherText 암호문 출력 버퍼 **반환값 :** 암호문의 Byte 길이 **참 고 :** 패딩 로직때문에 8바이트 블럭으로 처리함으로 pbszCipherText는 평문보다 8바이트 커야 한다. (평문이 8바이트 블럭 시 패딩 데이타가 8바이트가 들어간다.)

int HIGHT_CTR_Decrypt(IN BYTE *pbszUserKey, IN BYTE *pbszIV, IN BYTE *pbszCipherText, IN int nCipherTextLen, OUT BYTE *pbszPlainText)
HIGHT-CTR 알고리즘 복호화 함수(처리하고자 하는 데이터가 적을 경우에 사용) **매개변수 :** pbszUserKey 사용자가 지정하는 입력 키(16 BYTE) pszbIV 사용자가 지정하는 초기화 벡터(8 BYTE) pbszCipherText 암호문 nCipherTextLen 암호문 길이(BYTE 단위의 평문길이) pbszPlainText 평문 출력 버퍼 **반환값 :** 평문의 Byte 길이

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

HIGHT-CTR 알고리즘 초기화 함수

매개변수 :

pInfo	CBC 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(8 BYTE)

반환값 :

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

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

HIGHT-CTR 다중 블럭 암호화/복호화 함수

매개변수 :

pInfo	HIGHT_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 HIGHT_CTR_Close(OUT KISA_SEED_INFO *pInfo, IN DWORD *out, IN int *outLen)

HIGHT-CTR운영모드 종료 및 패딩 처리(PKCS7)

매개변수 :

pInfo	HIGHT_CTR_Process 를 거친 KISA_HIGHT_INFO 구조체
out	평문/암호문 출력 버퍼
outLen	출력 버퍼에 저장된 데이터 길이(BYTE 단위의 평문길이)

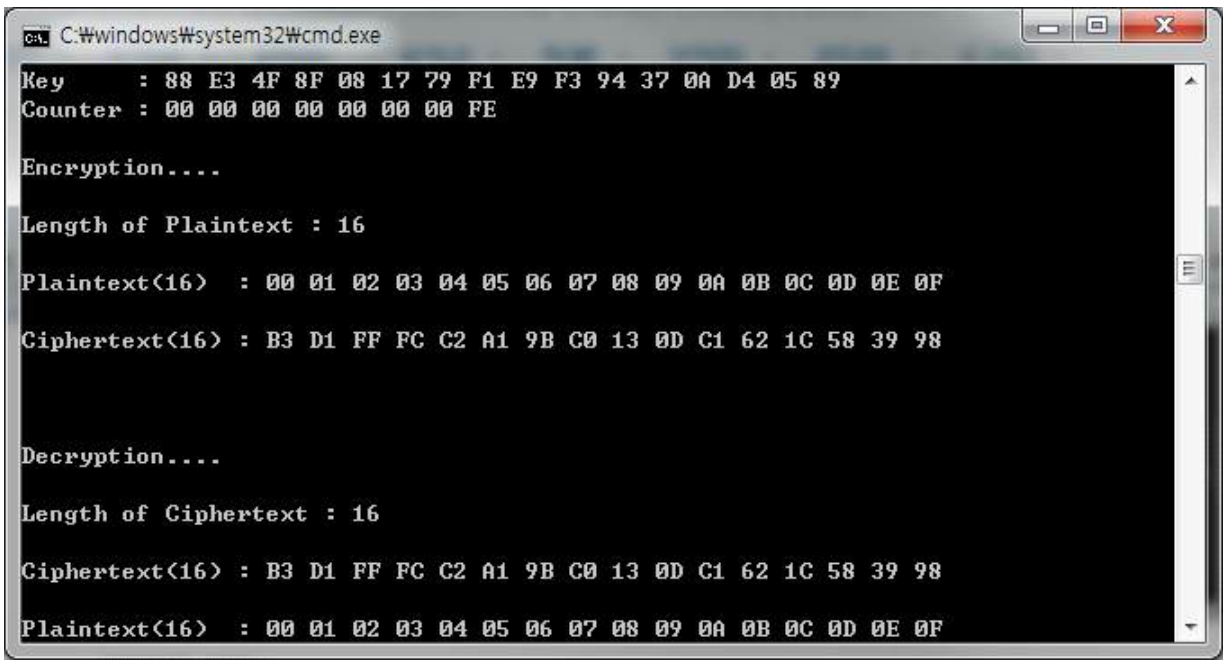
반환값 :

- 0 : inLen의 값이 0보다 작은 경우,
  KISA_SEED_INFO 구조체나 out에 널 포인터가 할당되었을 경우
- 1 : 성공

참 고 :

패딩 로직때문에 16바이트 블럭으로 처리함으로 복호화 시 출력 버퍼는 평문보다 16바이트 커야 한다.(평문이 16바이트 블럭 시 패딩 데이타가 16바이트가 들어간다.)

☞ **테스트 페이지**



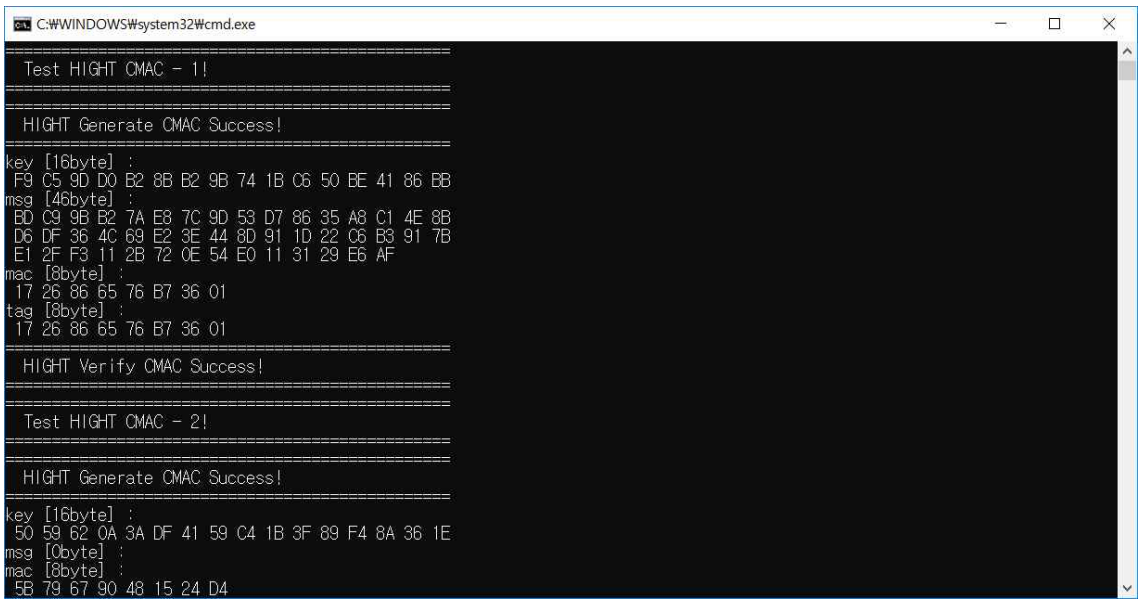
o HIGHT-CMAC

☞ 함수 설명

int HIGHT_Generate_CMAC(unsigned char *pMAC, int macLen, unsigned char *pIn, int inLen, unsigned char *mKey)
HIGHT-CMAC 알고리즘 MAC 생성 함수 **매개변수 :** pMAC MAC 출력 버퍼 macLen MAC 길이 pIn 사용자 입력 평문 inLen 평문 길이(BYTE 단위의 평문 길이) mKey 사용자가 지정하는 입력 키(16 BYTE) **반환값 :** - 0: MAC 생성 성공 - 1: MAC 길이가 블록 크기인 8BYTE 보다 큰 경우

int HIGHT_Verify_CMAC(unsigned char *pMAC, int macLen, unsigned char *pIn, int inLen, unsigned char *mKey)
HIGHT-CMAC 알고리즘 MAC 검증 함수 **매개변수 :** pMAC 검증할 MAC macLen MAC 길이 pIn 사용자 입력 평문 inLen 평문 길이(BYTE 단위의 평문 길이) mKey 사용자가 지정하는 입력 키(16 BYTE) **반환값 :** - 0: MAC 검증 성공 - 1: MAC 길이가 블록 크기인 8BYTE 보다 크거나 MAC 검증에 실패한 경우

☞ 테스트 페이지

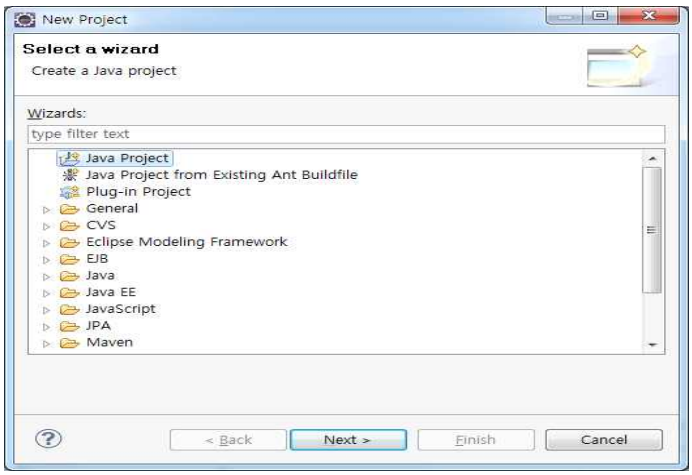

C:₩WINDOWS₩system32₩cmd.exe
Test HIGHT CMAC - 1!
HIGHT Generate CMAC Success!
key [16byte] :
F9 C5 9D D0 B2 8B B2 9B 74 1B C6 50 BE 41 86 BB
msg [46byte] :
BD C9 9B B2 7A E8 7C 9D 53 D7 86 35 A8 C1 4E 8B
D6 DF 36 4C 69 E2 3E 44 8D 91 1D 22 C6 B3 91 7B
E1 2F F3 11 2B 72 0E 54 E0 11 31 29 E6 AF
mac [8byte] :
17 26 86 65 76 B7 36 01
tag [8byte] :
17 26 86 65 76 B7 36 01
HIGHT Verify CMAC Success!
Test HIGHT CMAC - 2!
HIGHT Generate CMAC Success!
key [16byte] :
50 59 62 0A 3A DF 41 59 C4 1B 3F 89 F4 8A 36 1E
msg [0byte] :
mac [8byte] :
5B 79 67 90 48 15 24 D4

4.2. Java

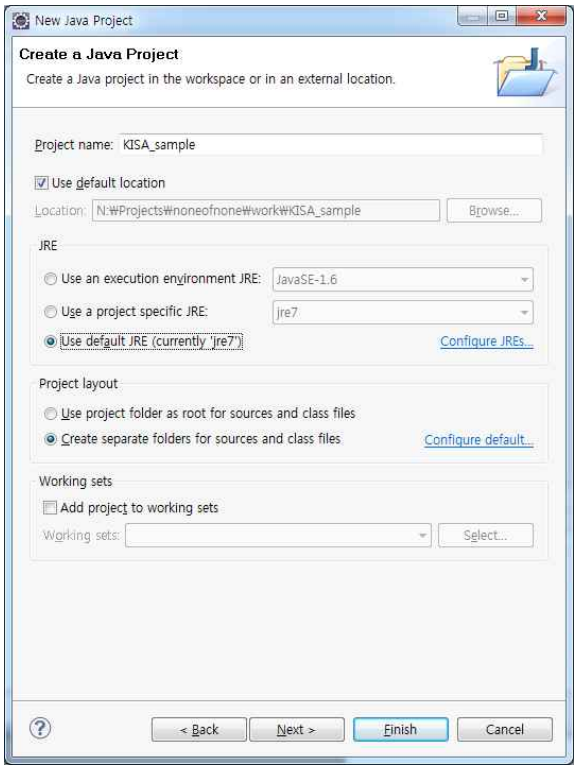
다음은 Java 형태의 프로그램을 하나 생성하여 헤더 및 소스와 파일 추가 하는 방법과 빌드하는 방법 등을 설명한다.

4.2.1 프로젝트 생성 및 빌드

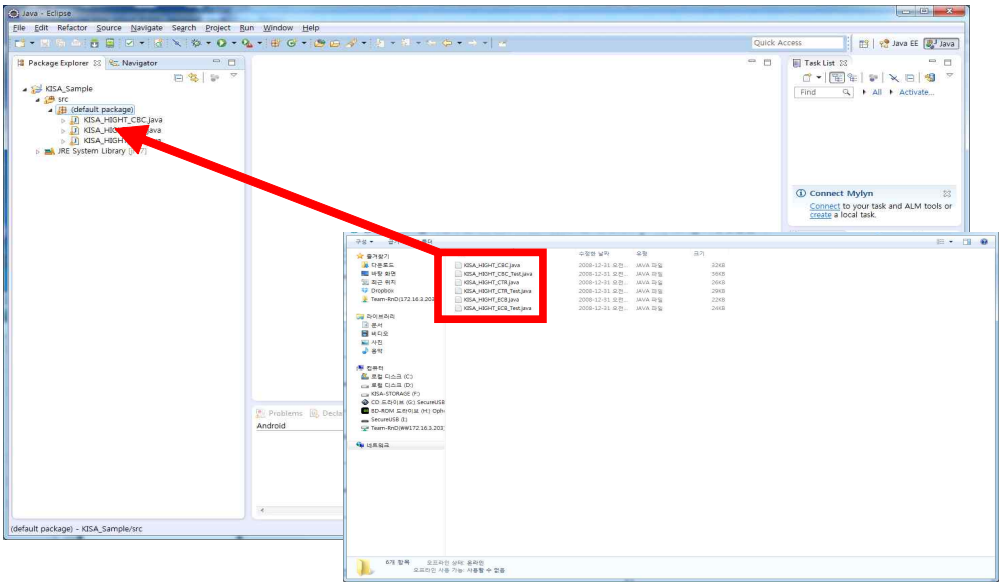
Eclipse를 실행하여 하기 이미지처럼 Java 프로젝트를 선택한다.



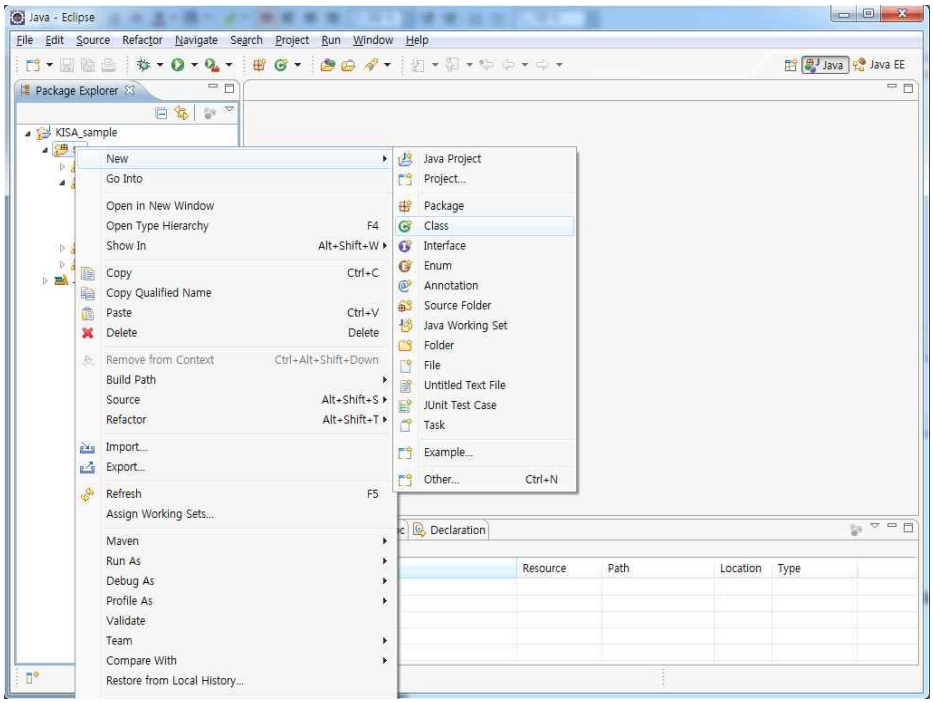
Next를 클릭하여 Project name에는 프로젝트 명을 기입한다. "Use default JRE" 를 선택 후 Finish 버튼을 눌러 프로젝트 생성을 완료한다.



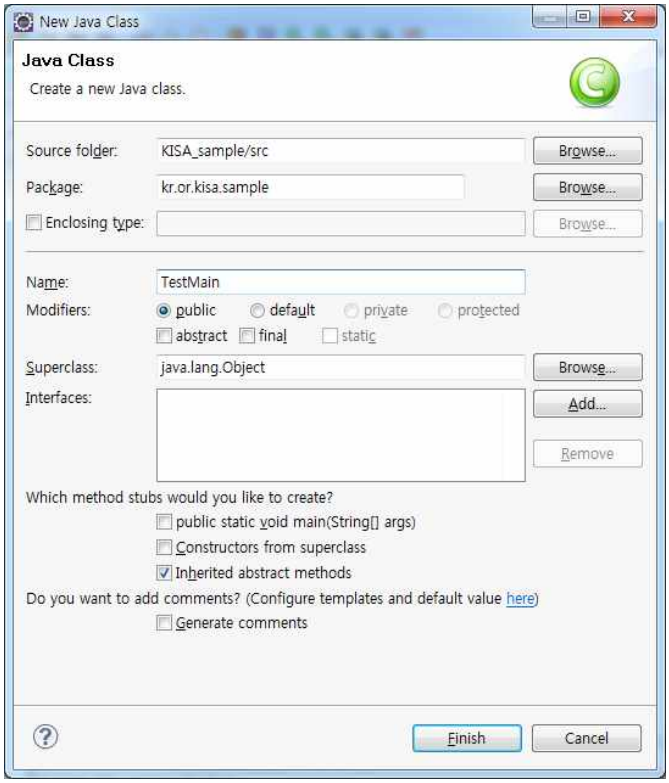
소스가 있는 폴더로 이동하여 프로젝트에 추가할 파일을 드레그&드롭으로 프로젝트로 이동시킨다. 이때 “Copy files and folders”를 선택하여 복사하여 준다.



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

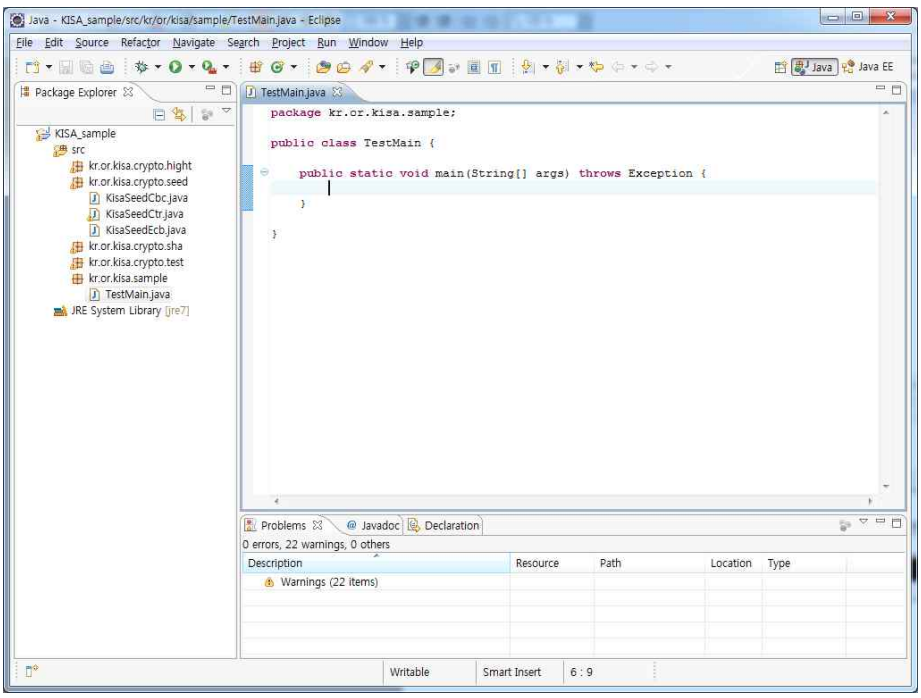


다음으로는 Package와 Name을 입력 후 Finish 버튼을 클릭하여 테스트 클래스를 생성한다.



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 HIGHT-ECB

☞ **함수 설명**

public static byte[] HIGHT_ECB_Encrypt(byte[] pbszUserKey, byte[] message, int message_offset, int message_length)
HIGHT-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 byte **참 고 :** 1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

public static byte[] HIGHT_ECB_Decrypt(byte[] pbszUserKey, byte[] message, int message_offset, int message_length)
HIGHT-ECB 알고리즘 복호화 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 평문 출력 byte **참 고 :** 1. pbszUserKey 의 크기는 반드시 16 bytes 여야 한다.

☞ 테스트 페이지

```
[ Test HIGHT 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

Ciphertext(HIGHT_ECB_Encrypt)   : e4 bc 2e 31 22 77 e4 dd 80 f2 12 8d 11 ac 4e 9a
Plaintext(HIGHT_ECB_Decrypt)    : d7 6d d 18 32 7e c5 62
```

o HIGHT-CBC

public static byte[] HIGHT_CBC_Encrypt(byte[] pbszUserKey, byte[] pszbIV, byte[] message, int message_offset, int message_length)

HIGHT-CBC 알고리즘 암호화 함수

매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (8 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 8 bytes 여야 한다.

public static byte[] HIGHT_CBC_Decrypt(byte[] pbszUserKey, byte[] pszbIV, byte[] message, int message_offset, int message_length)

HIGHT-CBC 알고리즘 복호화 함수

매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszIV	사용자가 지정하는 초기화 벡터 (8 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 하다.
2. pbszIV 의 크기는 반드시 8 bytes 여야 하다.

public static int HIGHT_CBC_init(KISA_HIGHT_INFO pInfo, KISA_ENC_DEC enc, byte[] pUserKey, byte[] pbszIV)

HIGHT-CBC 알고리즘 초기화 함수

매개변수 :

pInfo	HIGHT 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 (8 bytes)

반환값 :

- 1 : 초기화 성공
- 0 : 초기화 실패

public static int HIGHT_CBC_Process(KISA_HIGHT_INFO pInfo, int[] in, int inLen, int[] out, int[] outLen)

HIGHT-CBC 알고리즘 다중 블록 암호화 함수

매개변수 :

pInfo	HIGHT CBC 알고리즘 운영을 위한 클래스 지정 (HIGHT_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 HIGHT_CBC_Close(KISA_HIGHT_INFO pInfo, int[] out, int out_offset, int[] outLen)
HIGHT-CBC 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수 **매개변수 :** pInfo HIGHT CBC 알고리즘 운영을 위한 클래스 지정 (HIGHT_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 HIGHT 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
IV                               : 26 8d 66 a7 35 a8 1a 81

Ciphertext(HIGHT_CBC_Encrypt)    : 9c, 8f, a0, a5, 9f, 9e, 36, 31,
Plaintext(HIGHT_CBC_Decrypt)     : d7, 6d, d, 18, 32, 7e, c5, 62,
```

o HIGHT-CTR

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

HIGHT-CTR 알고리즘 암호화 함수

매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (8 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 8 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼의 크기와 동일(패딩 처리를 하지 않는다.)

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

HIGHT-CTR 알고리즘 복호화 함수

매개변수 :

pbszUserKey	사용자가 지정하는 입력 키 (16 bytes)
pbszCTR	사용자가 지정하는 초기화 벡터 (8 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 8 bytes 여야 한다.
3. 출력 버퍼의 크기는 입력 버퍼 크기와 동일하다.

public static void HIGHT_CTR_init(KISA_HIGHT_INFO pInfo, KISA_ENC_DEC enc, byte[] pUserKey, byte[] pbszCTR)

HIGHT-CTR 알고리즘 초기화 함수

매개변수 :

pInfo	HIGHT 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 (8 bytes)

public static int HIGHT_CTR_Process(KISA_HIGHT_INFO pInfo, int[] in, int inLen, int[] out, int[] outLen)

HIGHT-CTR 알고리즘 다중 블록 암호화 함수

매개변수 :

pInfo	HIGHT CTR 알고리즘 운영을 위한 클래스 지정 (HIGHT_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 HIGHT_CTR_Close(KISA_HIGHT_INFO pInfo, int[] out, int out_offset, int[] outLen)
HIGHT-CTR 알고리즘 운영모드 종료 및 패딩(PKCS7) 처리 함수 **매개변수 :** pInfo HIGHT CTR 알고리즘 운영을 위한 클래스 지정 (HIGHT_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 HIGHT 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
CTR                             : 0 0 0 0 0 0 0 fe

Ciphertext(HIGHT_CTR_Encrypt)   : 64 bd f0 e7 f4 da 58 a5
Plaintext(HIGHT_CTR_Decrypt)    : d7 6d d 18 32 7e c5 62
```

o HIGHT-CMAC

☞ 함수 설명

public int HIGHT_Generate_CMAC(byte[] pMAC, int macLen, byte[] pIn, int inLen, byte[] mKey)
HIGHT-CMAC 알고리즘 MAC 생성 함수 **매개변수 :** pMAC MAC 출력 버퍼 macLen MAC 길이 pIn 사용자 입력 평문 inLen 평문 길이(BYTE 단위의 평문 길이) mKey 사용자가 지정하는 입력 키(16 BYTE) **반환값 :** - 0: MAC 생성 성공 - 1: MAC 길이가 블록 크기인 8BYTE 보다 큰 경우

public int HIGHT_Verify_CMAC(byte[] pMAC, int macLen, byte[] pIn, int inLen, byte[] mKey)
HIGHT-CMAC 알고리즘 MAC 검증 함수 **매개변수 :** pMAC 검증할 MAC macLen MAC 길이 pIn 사용자 입력 평문 inLen 평문 길이(BYTE 단위의 평문 길이) mKey 사용자가 지정하는 입력 키(16 BYTE) **반환값 :** - 0: MAC 검증 성공 - 1: MAC 길이가 블록 크기인 8BYTE 보다 크거나 MAC 검증에 실패한 경우

☞ 테스트 페이지

```
================================================
  Test HIGHT CMAC - 1
================================================
================================================
  HIGHT Generate CMAC Success!
================================================
key [16byte] :
 F9 C5 9D D0 B2 8B B2 9B 74 1B C6 50 BE 41 86 BB
msg [46byte] :
 BD C9 9B B2 7A E8 7C 9D 53 D7 86 35 A8 C1 4E 8B
 D6 DF 36 4C 69 E2 3E 44 8D 91 1D 22 C6 B3 91 7B
 E1 2F F3 11 2B 72 0E 54 E0 11 31 29 E6 AF
mac [8byte] :
 17 26 86 65 76 B7 36 01
tag [8byte] :
 17 26 86 65 76 B7 36 01
================================================
  HIGHT Verify CMAC Success!
================================================
```

[부록]
참조구현값

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

ECB, CBC, CTR 운영모드에 대한 참조구현값을 생성하기 위한 평문 데이터와 키, 초기값, 초기 카운트는 아래 표와 같다. 데이터 1, 2에 대한 참조구현값은 암·복호화 연산에서 단계별 입력 블록(I_i), 출력 블록(O_i)의 구체적인 중간값(Intermediate Value)이 추가로 제공된다.

< 데이터 1 >

키(Key) 1		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
평문(P) 1	블록1	D7 6D 0D 18 32 7E C5 62
	블록2	B1 5E 6B C3 65 AC 0C 0F
	블록3	8D 41 E0 BB 93 85 68 AE
	블록4	EB FD 92 ED 1A FF A0 96
	블록5	39 4D 20 FC 52 77 DD FC
	블록6	4D E8 B0 FC E1 EB 2B 93
	블록7	D4 AE 40 EF 47 68 C6 13
	블록8	B5 0B 89 42 F7 D4 B9 B3

< 데이터 2 >

키(Key) 2		2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C
초기값(IV)		26 8D 66 A7 35 A8 1A 81
평문(P) 2	블록1	6B C1 BE E2 2E 40 9F 96
	블록2	E9 3D 7E 11 73 93 17 2A
	블록3	AE 2D 8A 57 1E 03 AC 9C
	블록4	9E B7 6F AC 45 AF 8E 51
	블록5	30 C8 1C 46 A3 5C E4 11
	블록6	E5 FB C1 19 1A 0A 52 EF
	블록7	F6 9F 24 45 DF 4F 9B 17
	블록8	AD 2B 41 7B E6 6C 37 10

CMAC 운영모드에 대한 참조구현값을 생성하기 위한 평문 데이터, 키는 아래 표와 같다. 데이터 3, 4에 대한 참조구현값은 평문 데이터를 이용하여 MAC값을 생성하기 위한 중간값들을 제공한다.

< 데이터 3 >

키(Key)	F9 C5 9D D0 B2 8B B2 9B 74 1B C6 50 BE 41 86 BB
평문(P)	BD C9 9B B2 7A E8 7C 9D 53 D7 86 35 A8 C1 4E 8B D6 DF 36 4C 69 E2 3E 44 8D 91 1D 22 C6 B3 91 7B E1 2F F3 11 2B 72 0E 54 E0 11 31 29 E6 AF
MAC 길이	8 byte

< 데이터 4 >

키(Key)	50 59 62 0A 3A DF 41 59 C4 1B 3F 89 F4 8A 36 1E
평문(P)	(NULL)
MAC 길이	8 byte

1. ECB 모드 참조구현값

1.1 데이터 1 - 암호화

블록1	평문(P1)	D7 6D 0D 18 32 7E C5 62
	입력(I1)	D7 6D 0D 18 32 7E C5 62
	출력(O1)	E4 BC 2E 31 22 77 E4 DD
	암호문(C1)	E4 BC 2E 31 22 77 E4 DD
블록2	평문(P2)	B1 5E 6B C3 65 AC 0C 0F
	입력(I2)	B1 5E 6B C3 65 AC 0C 0F
	출력(O2)	A0 14 7A FB AC 9D 28 99
	암호문(C2)	A0 14 7A FB AC 9D 28 99
블록3	평문(P3)	8D 41 E0 BB 93 85 68 AE
	입력(I3)	8D 41 E0 BB 93 85 68 AE
	출력(O3)	9D 76 E8 06 78 F9 85 1C
	암호문(C3)	9D 76 E8 06 78 F9 85 1C
블록4	평문(P4)	EB FD 92 ED 1A FF A0 96
	입력(I4)	EB FD 92 ED 1A FF A0 96
	출력(O4)	27 4C 1B 4D AF 76 9B AA
	암호문(C4)	27 4C 1B 4D AF 76 9B AA
블록5	평문(P5)	39 4D 20 FC 52 77 DD FC
	입력(I5)	39 4D 20 FC 52 77 DD FC
	출력(O5)	1C 1D 73 23 42 70 F0 B0
	암호문(C5)	1C 1D 73 23 42 70 F0 B0
블록6	평문(P6)	4D E8 B0 FC E1 EB 2B 93
	입력(I6)	4D E8 B0 FC E1 EB 2B 93
	출력(O6)	09 5A 14 54 E1 92 AD DD
	암호문(C6)	09 5A 14 54 E1 92 AD DD
블록7	평문(P7)	D4 AE 40 EF 47 68 C6 13
	입력(I7)	D4 AE 40 EF 47 68 C6 13
	출력(O7)	3C 9E 22 A4 ED 61 5C 31
	암호문(C7)	3C 9E 22 A4 ED 61 5C 31
블록8	평문(P8)	B5 0B 89 42 F7 D4 B9 B3
	입력(I8)	B5 0B 89 42 F7 D4 B9 B3
	출력(O8)	17 5E 90 FB E7 3A 55 08
	암호문(C8)	17 5E 90 FB E7 3A 55 08

1.2 데이터 1 - 복호화

블록1	암호문(C1)	E4 BC 2E 31 22 77 E4 DD
	입력(I1)	E4 BC 2E 31 22 77 E4 DD
	출력(O1)	D7 6D 0D 18 32 7E C5 62
	평문(P1)	D7 6D 0D 18 32 7E C5 62
블록2	암호문(C2)	A0 14 7A FB AC 9D 28 99
	입력(I2)	A0 14 7A FB AC 9D 28 99
	출력(O2)	B1 5E 6B C3 65 AC 0C 0F
	평문(P2)	B1 5E 6B C3 65 AC 0C 0F
블록3	암호문(C3)	9D 76 E8 06 78 F9 85 1C
	입력(I3)	9D 76 E8 06 78 F9 85 1C
	출력(O3)	8D 41 E0 BB 93 85 68 AE
	평문(P3)	8D 41 E0 BB 93 85 68 AE
블록4	암호문(C4)	27 4C 1B 4D AF 76 9B AA
	입력(I4)	27 4C 1B 4D AF 76 9B AA
	출력(O4)	EB FD 92 ED 1A FF A0 96
	평문(P4)	EB FD 92 ED 1A FF A0 96
블록5	암호문(C5)	1C 1D 73 23 42 70 F0 B0
	입력(I5)	1C 1D 73 23 42 70 F0 B0
	출력(O5)	39 4D 20 FC 52 77 DD FC
	평문(P5)	39 4D 20 FC 52 77 DD FC
블록6	암호문(C6)	09 5A 14 54 E1 92 AD DD
	입력(I6)	09 5A 14 54 E1 92 AD DD
	출력(O6)	4D E8 B0 FC E1 EB 2B 93
	평문(P6)	4D E8 B0 FC E1 EB 2B 93
블록7	암호문(C7)	3C 9E 22 A4 ED 61 5C 31
	입력(I7)	3C 9E 22 A4 ED 61 5C 31
	출력(O7)	D4 AE 40 EF 47 68 C6 13
	평문(P7)	D4 AE 40 EF 47 68 C6 13
블록8	암호문(C8)	17 5E 90 FB E7 3A 55 08
	입력(I8)	17 5E 90 FB E7 3A 55 08
	출력(O8)	B5 0B 89 42 F7 D4 B9 B3
	평문(P8)	B5 0B 89 42 F7 D4 B9 B3

1.3 데이터 2 - 암호화

블록1	평문(P1)	6B C1 BE E2 2E 40 9F 96
	입력(I1)	6B C1 BE E2 2E 40 9F 96
	출력(O1)	98 13 D3 2C E7 FD 5A BB
	암호문(C1)	98 13 D3 2C E7 FD 5A BB
블록2	평문(P2)	E9 3D 7E 11 73 93 17 2A
	입력(I2)	E9 3D 7E 11 73 93 17 2A
	출력(O2)	01 13 B3 2D 34 E6 24 3F
	암호문(C2)	01 13 B3 2D 34 E6 24 3F
블록3	평문(P3)	AE 2D 8A 57 1E 03 AC 9C
	입력(I3)	AE 2D 8A 57 1E 03 AC 9C
	출력(O3)	95 EB A8 45 88 A7 0B C7
	암호문(C3)	95 EB A8 45 88 A7 0B C7
블록4	평문(P4)	9E B7 6F AC 45 AF 8E 51
	입력(I4)	9E B7 6F AC 45 AF 8E 51
	출력(O4)	03 0B D8 79 1A 35 62 5B
	암호문(C4)	03 0B D8 79 1A 35 62 5B
블록5	평문(P5)	30 C8 1C 46 A3 5C E4 11
	입력(I5)	30 C8 1C 46 A3 5C E4 11
	출력(O5)	DC 87 3B 51 75 D4 BF 97
	암호문(C5)	DC 87 3B 51 75 D4 BF 97
블록6	평문(P6)	E5 FB C1 19 1A 0A 52 EF
	입력(I6)	E5 FB C1 19 1A 0A 52 EF
	출력(O6)	7A 65 51 49 3A B2 63 F2
	암호문(C6)	7A 65 51 49 3A B2 63 F2
블록7	평문(P7)	F6 9F 24 45 DF 4F 9B 17
	입력(I7)	F6 9F 24 45 DF 4F 9B 17
	출력(O7)	6D 20 C1 EE 2C 6C 31 A3
	암호문(C7)	6D 20 C1 EE 2C 6C 31 A3
블록8	평문(P8)	AD 2B 41 7B E6 6C 37 10
	입력(I8)	AD 2B 41 7B E6 6C 37 10
	출력(O8)	12 F6 CE E2 FE 47 59 A5
	암호문(C8)	12 F6 CE E2 FE 47 59 A5

1.4 데이터 2 - 복호화

블록1	암호문(C1)	98 13 D3 2C E7 FD 5A BB
	입력(I1)	98 13 D3 2C E7 FD 5A BB
	출력(O1)	6B C1 BE E2 2E 40 9F 96
	평문(P1)	6B C1 BE E2 2E 40 9F 96
블록2	암호문(C2)	01 13 B3 2D 34 E6 24 3F
	입력(I2)	01 13 B3 2D 34 E6 24 3F
	출력(O2)	E9 3D 7E 11 73 93 17 2A
	평문(P2)	E9 3D 7E 11 73 93 17 2A
블록3	암호문(C3)	95 EB A8 45 88 A7 0B C7
	입력(I3)	95 EB A8 45 88 A7 0B C7
	출력(O3)	AE 2D 8A 57 1E 03 AC 9C
	평문(P3)	AE 2D 8A 57 1E 03 AC 9C
블록4	암호문(C4)	03 0B D8 79 1A 35 62 5B
	입력(I4)	03 0B D8 79 1A 35 62 5B
	출력(O4)	9E B7 6F AC 45 AF 8E 51
	평문(P4)	9E B7 6F AC 45 AF 8E 51
블록5	암호문(C5)	DC 87 3B 51 75 D4 BF 97
	입력(I5)	DC 87 3B 51 75 D4 BF 97
	출력(O5)	30 C8 1C 46 A3 5C E4 11
	평문(P5)	30 C8 1C 46 A3 5C E4 11
블록6	암호문(C6)	7A 65 51 49 3A B2 63 F2
	입력(I6)	7A 65 51 49 3A B2 63 F2
	출력(O6)	E5 FB C1 19 1A 0A 52 EF
	평문(P6)	E5 FB C1 19 1A 0A 52 EF
블록7	암호문(C7)	6D 20 C1 EE 2C 6C 31 A3
	입력(I7)	6D 20 C1 EE 2C 6C 31 A3
	출력(O7)	F6 9F 24 45 DF 4F 9B 17
	평문(P7)	F6 9F 24 45 DF 4F 9B 17
블록8	암호문(C8)	12 F6 CE E2 FE 47 59 A5
	입력(I8)	12 F6 CE E2 FE 47 59 A5
	출력(O8)	AD 2B 41 7B E6 6C 37 10
	평문(P8)	AD 2B 41 7B E6 6C 37 10

2. CBC 모드 참조구현값

2.1 데이터 1 - 암호화

블록1	평문(P1)	D7 6D 0D 18 32 7E C5 62
	입력(I1)	F1 E0 6B BF 07 D6 DF E3
	출력(O1)	9C 8F A0 A5 9F 9E 36 31
	암호문(C1)	9C 8F A0 A5 9F 9E 36 31
블록2	평문(P2)	B1 5E 6B C3 65 AC 0C 0F
	입력(I2)	2D D1 CB 66 FA 32 3A 3E
	출력(O2)	A6 E7 CB D3 C4 24 26 B8
	암호문(C2)	A6 E7 CB D3 C4 24 26 B8
블록3	평문(P3)	8D 41 E0 BB 93 85 68 AE
	입력(I3)	2B A6 2B 68 57 A1 4E 16
	출력(O3)	1F 12 06 12 A4 0E 43 AD
	암호문(C3)	1F 12 06 12 A4 0E 43 AD
블록4	평문(P4)	EB FD 92 ED 1A FF A0 96
	입력(I4)	F4 EF 94 FF BE F1 E3 3B
	출력(O4)	78 4F 42 26 A3 71 44 63
	암호문(C4)	78 4F 42 26 A3 71 44 63
블록5	평문(P5)	39 4D 20 FC 52 77 DD FC
	입력(I5)	41 02 62 DA F1 06 99 9F
	출력(O5)	DA 5F B1 C3 C0 D8 28 CF
	암호문(C5)	DA 5F B1 C3 C0 D8 28 CF
블록6	평문(P6)	4D E8 B0 FC E1 EB 2B 93
	입력(I6)	97 B7 01 3F 21 33 03 5C
	출력(O6)	18 E0 48 03 A1 B7 9C 43
	암호문(C6)	18 E0 48 03 A1 B7 9C 43
블록7	평문(P7)	D4 AE 40 EF 47 68 C6 13
	입력(I7)	CC 4E 08 CC E6 DF 5A 50
	출력(O7)	4C D5 8B 74 C5 85 ED 18
	암호문(C7)	4C D5 8B 74 C5 85 ED 18
블록8	평문(P8)	B5 0B 89 42 F7 D4 B9 B3
	입력(I8)	F9 DE 02 36 32 51 54 AB
	출력(O8)	D1 AB 55 E0 4A AB E7 65
	암호문(C8)	D1 AB 55 E0 4A AB E7 65

2.2 데이터 1 - 복호화

블록1	암호문(C1)	9C 8F A0 A5 9F 9E 36 31
	입력(I1)	9C 8F A0 A5 9F 9E 36 31
	출력(O1)	F1 E0 6B BF 07 D6 DF E3
	평문(P1)	D7 6D 0D 18 32 7E C5 62
블록2	암호문(C2)	A6 E7 CB D3 C4 24 26 B8
	입력(I2)	A6 E7 CB D3 C4 24 26 B8
	출력(O2)	2D D1 CB 66 FA 32 3A 3E
	평문(P2)	B1 5E 6B C3 65 AC 0C 0F
블록3	암호문(C3)	1F 12 06 12 A4 0E 43 AD
	입력(I3)	1F 12 06 12 A4 0E 43 AD
	출력(O3)	2B A6 2B 68 57 A1 4E 16
	평문(P3)	8D 41 E0 BB 93 85 68 AE
블록4	암호문(C4)	78 4F 42 26 A3 71 44 63
	입력(I4)	78 4F 42 26 A3 71 44 63
	출력(O4)	F4 EF 94 FF BE F1 E3 3B
	평문(P4)	EB FD 92 ED 1A FF A0 96
블록5	암호문(C5)	DA 5F B1 C3 C0 D8 28 CF
	입력(I5)	DA 5F B1 C3 C0 D8 28 CF
	출력(O5)	41 02 62 DA F1 06 99 9F
	평문(P5)	39 4D 20 FC 52 77 DD FC
블록6	암호문(C6)	18 E0 48 03 A1 B7 9C 43
	입력(I6)	18 E0 48 03 A1 B7 9C 43
	출력(O6)	97 B7 01 3F 21 33 03 5C
	평문(P6)	4D E8 B0 FC E1 EB 2B 93
블록7	암호문(C7)	4C D5 8B 74 C5 85 ED 18
	입력(I7)	4C D5 8B 74 C5 85 ED 18
	출력(O7)	CC 4E 08 CC E6 DF 5A 50
	평문(P7)	D4 AE 40 EF 47 68 C6 13
블록8	암호문(C8)	D1 AB 55 E0 4A AB E7 65
	입력(I8)	D1 AB 55 E0 4A AB E7 65
	출력(O8)	F9 DE 02 36 32 51 54 AB
	평문(P8)	B5 0B 89 42 F7 D4 B9 B3

2.3 데이터 2 - 암호화

블록1	평문(P1)	6B C1 BE E2 2E 40 9F 96
	입력(I1)	4D 4C D8 45 1B E8 85 17
	출력(O1)	89 9A C0 10 90 3A 53 FD
	암호문(C1)	89 9A C0 10 90 3A 53 FD
블록2	평문(P2)	E9 3D 7E 11 73 93 17 2A
	입력(I2)	60 A7 BE 01 E3 A9 44 D7
	출력(O2)	EA 8D AB 03 76 32 5E 87
	암호문(C2)	EA 8D AB 03 76 32 5E 87
블록3	평문(P3)	AE 2D 8A 57 1E 03 AC 9C
	입력(I3)	44 A0 21 54 68 31 F2 1B
	출력(O3)	25 D3 68 1E 04 9F 32 3F
	암호문(C3)	25 D3 68 1E 04 9F 32 3F
블록4	평문(P4)	9E B7 6F AC 45 AF 8E 51
	입력(I4)	BB 64 07 B2 41 30 BC 6E
	출력(O4)	DA AC DE 16 4A 02 BB 28
	암호문(C4)	DA AC DE 16 4A 02 BB 28
블록5	평문(P5)	30 C8 1C 46 A3 5C E4 11
	입력(I5)	EA 64 C2 50 E9 5E 5F 39
	출력(O5)	1A 17 E3 D8 13 D4 0B 06
	암호문(C5)	1A 17 E3 D8 13 D4 0B 06
블록6	평문(P6)	E5 FB C1 19 1A 0A 52 EF
	입력(I6)	FF EC 22 C1 09 DE 59 E9
	출력(O6)	A0 C9 F4 15 E0 60 AD 5F
	암호문(C6)	A0 C9 F4 15 E0 60 AD 5F
블록7	평문(P7)	F6 9F 24 45 DF 4F 9B 17
	입력(I7)	56 56 D0 50 3F 2F 36 48
	출력(O7)	17 33 EF 74 EC DB 7C B7
	암호문(C7)	17 33 EF 74 EC DB 7C B7
블록8	평문(P8)	AD 2B 41 7B E6 6C 37 10
	입력(I8)	BA 18 AE 0F 0A B7 4B A7
	출력(O8)	DC 01 A6 5F 5C 8A E3 49
	암호문(C8)	DC 01 A6 5F 5C 8A E3 49

2.4 데이터 2 - 복호화

블록1	암호문(C1)	89 9A C0 10 90 3A 53 FD
	입력(I1)	89 9A C0 10 90 3A 53 FD
	출력(O1)	4D 4C D8 45 1B E8 85 17
	평문(P1)	6B C1 BE E2 2E 40 9F 96
블록2	암호문(C2)	EA 8D AB 03 76 32 5E 87
	입력(I2)	EA 8D AB 03 76 32 5E 87
	출력(O2)	60 A7 BE 01 E3 A9 44 D7
	평문(P2)	E9 3D 7E 11 73 93 17 2A
블록3	암호문(C3)	25 D3 68 1E 04 9F 32 3F
	입력(I3)	25 D3 68 1E 04 9F 32 3F
	출력(O3)	44 A0 21 54 68 31 F2 1B
	평문(P3)	AE 2D 8A 57 1E 03 AC 9C
블록4	암호문(C4)	DA AC DE 16 4A 02 BB 28
	입력(I4)	DA AC DE 16 4A 02 BB 28
	출력(O4)	BB 64 07 B2 41 30 BC 6E
	평문(P4)	9E B7 6F AC 45 AF 8E 51
블록5	암호문(C5)	1A 17 E3 D8 13 D4 0B 06
	입력(I5)	1A 17 E3 D8 13 D4 0B 06
	출력(O5)	EA 64 C2 50 E9 5E 5F 39
	평문(P5)	30 C8 1C 46 A3 5C E4 11
블록6	암호문(C6)	A0 C9 F4 15 E0 60 AD 5F
	입력(I6)	A0 C9 F4 15 E0 60 AD 5F
	출력(O6)	FF EC 22 C1 09 DE 59 E9
	평문(P6)	E5 FB C1 19 1A 0A 52 EF
블록7	암호문(C7)	17 33 EF 74 EC DB 7C B7
	입력(I7)	17 33 EF 74 EC DB 7C B7
	출력(O7)	56 56 D0 50 3F 2F 36 48
	평문(P7)	F6 9F 24 45 DF 4F 9B 17
블록8	암호문(C8)	DC 01 A6 5F 5C 8A E3 49
	입력(I8)	DC 01 A6 5F 5C 8A E3 49
	출력(O8)	BA 18 AE 0F 0A B7 4B A7
	평문(P8)	AD 2B 41 7B E6 6C 37 10

3. CTR 모드 참조구현값

3.1 데이터 1 - 암호화

블록1	입력(P1)	00 00 00 00 00 00 00 FE
	출력(I1)	b3 d0 fd ff c6 a4 9d c7
	평문(01)	D7 6D 0D 18 32 7E C5 62
	암호문(C1)	64 bd f0 e7 f4 da 58 a5
블록2	입력(P2)	00 00 00 00 00 00 00 FF
	출력(I2)	1B 04 CB 69 10 55 37 97
	평문(02)	B1 5E 6B C3 65 AC 0C 0F
	암호문(C2)	AA 5A A0 AA 75 F9 3B 98
블록3	입력(P3)	00 00 00 00 00 00 01 00
	출력(I3)	8A D6 C7 98 44 A7 D3 BE
	평문(03)	8D 41 E0 BB 93 85 68 AE
	암호문(C3)	07 97 27 23 D7 22 BB 10
블록4	입력(P4)	00 00 00 00 00 00 01 01
	출력(I4)	5F 73 D5 02 BA AC 92 AC
	평문(04)	EB FD 92 ED 1A FF A0 96
	암호문(C4)	B4 8E 47 EF A0 53 32 3A
블록5	입력(P5)	00 00 00 00 00 00 01 02
	출력(I5)	D7 6B 47 38 F3 0E 0D 6B
	평문(05)	39 4D 20 FC 52 77 DD FC
	암호문(C5)	EE 26 67 C4 A1 79 D0 97
블록6	입력(P6)	00 00 00 00 00 00 01 03
	출력(I6)	70 47 5B BC 64 DE 12 E0
	평문(06)	4D E8 B0 FC E1 EB 2B 93
	암호문(C6)	3D AF EB 40 85 35 39 73
블록7	입력(P7)	00 00 00 00 00 00 01 04
	출력(I7)	B3 5F F5 F9 5B 02 5D 0E
	평문(07)	D4 AE 40 EF 47 68 C6 13
	암호문(C7)	67 F1 B5 16 1C 6A 9B 1D
블록8	입력(P8)	00 00 00 00 00 00 01 05
	출력(I8)	97 69 43 15 6C 92 CC 1C
	평문(08)	B5 0B 89 42 F7 D4 B9 B3
	암호문(C8)	22 62 CA 57 9B 46 75 AF

3.2 데이터 1 - 복호화

블록1	입력(P1)	00 00 00 00 00 00 00 FE
	출력(I1)	b3 d0 fd ff c6 a4 9d c7
	암호문(C1)	64 bd f0 e7 f4 da 58 a5
	평문(01)	D7 6D 0D 18 32 7E C5 62
블록2	입력(P2)	00 00 00 00 00 00 00 FF
	출력(I2)	1B 04 CB 69 10 55 37 97
	암호문(C2)	AA 5A A0 AA 75 F9 3B 98
	평문(02)	B1 5E 6B C3 65 AC 0C 0F
블록3	입력(P3)	00 00 00 00 00 00 01 00
	출력(I3)	8A D6 C7 98 44 A7 D3 BE
	암호문(C3)	07 97 27 23 D7 22 BB 10
	평문(03)	8D 41 E0 BB 93 85 68 AE
블록4	입력(P4)	00 00 00 00 00 00 01 01
	출력(I4)	5F 73 D5 02 BA AC 92 AC
	암호문(C4)	B4 8E 47 EF A0 53 32 3A
	평문(04)	EB FD 92 ED 1A FF A0 96
블록5	입력(P5)	00 00 00 00 00 00 01 02
	출력(I5)	D7 6B 47 38 F3 0E 0D 6B
	암호문(C5)	EE 26 67 C4 A1 79 D0 97
	평문(05)	39 4D 20 FC 52 77 DD FC
블록6	입력(P6)	00 00 00 00 00 00 01 03
	출력(I6)	70 47 5B BC 64 DE 12 E0
	암호문(C6)	3D AF EB 40 85 35 39 73
	평문(06)	4D E8 B0 FC E1 EB 2B 93
블록7	입력(P7)	00 00 00 00 00 00 01 04
	출력(I7)	B3 5F F5 F9 5B 02 5D 0E
	암호문(C7)	67 F1 B5 16 1C 6A 9B 1D
	평문(07)	D4 AE 40 EF 47 68 C6 13
블록8	입력(P8)	00 00 00 00 00 00 01 05
	출력(I8)	97 69 43 15 6C 92 CC 1C
	암호문(C8)	22 62 CA 57 9B 46 75 AF
	평문(08)	B5 0B 89 42 F7 D4 B9 B3

3.3 데이터 2 - 암호화

블록1	입력(P1)	00 00 00 00 00 00 00 FE
	출력(I1)	D8 75 17 60 9A 04 48 18
	평문(01)	6B C1 BE E2 2E 40 9F 96
	암호문(C1)	B3 B4 A9 82 B4 44 D7 8E
블록2	입력(P2)	00 00 00 00 00 00 00 FF
	출력(I2)	6D 55 EA 20 D9 6D 9A A9
	평문(02)	E9 3D 7E 11 73 93 17 2A
	암호문(C2)	84 68 94 31 AA FE 8D 83
블록3	입력(P3)	00 00 00 00 00 00 01 00
	출력(I3)	30 02 F7 15 F9 EB DF 1F
	평문(03)	AE 2D 8A 57 1E 03 AC 9C
	암호문(C3)	9E 2F 7D 42 E7 E8 73 83
블록4	입력(P4)	00 00 00 00 00 00 01 01
	출력(I4)	E1 82 63 86 97 7D 7C 3F
	평문(04)	9E B7 6F AC 45 AF 8E 51
	암호문(C4)	7F 35 0C 2A D2 D2 F2 6E
블록5	입력(P5)	00 00 00 00 00 00 01 02
	출력(I5)	83 EA 79 8E 46 9D 83 f0
	평문(05)	30 C8 1C 46 A3 5C E4 11
	암호문(C5)	B3 22 65 C8 E5 C1 67 E1
블록6	입력(P6)	00 00 00 00 00 00 01 03
	출력(I6)	80 EB 00 4E 35 08 E6 f6
	평문(06)	E5 FB C1 19 1A 0A 52 EF
	암호문(C6)	65 10 C1 57 2F 02 B4 19
블록7	입력(P7)	00 00 00 00 00 00 01 04
	출력(I7)	F2 88 89 17 50 B9 7D 97
	평문(07)	F6 9F 24 45 DF 4F 9B 17
	암호문(C7)	04 17 AD 52 8F F6 E6 80
블록8	입력(P8)	00 00 00 00 00 00 01 05
	출력(I8)	8B 10 86 EA AD 0C 41 BB
	평문(08)	AD 2B 41 7B E6 6C 37 10
	암호문(C8)	26 3B C7 91 4B 60 76 AB

3.4 데이터 2 - 복호화

블록1	입력(P1)	00 00 00 00 00 00 00 FE
	출력(I1)	D8 75 17 60 9A 04 48 18
	암호문(C1)	B3 B4 A9 82 B4 44 D7 8E
	평문(01)	6B C1 BE E2 2E 40 9F 96
블록2	입력(P2)	00 00 00 00 00 00 00 FF
	출력(I2)	6D 55 EA 20 D9 6D 9A A9
	암호문(C2)	84 68 94 31 AA FE 8D 83
	평문(02)	E9 3D 7E 11 73 93 17 2A
블록3	입력(P3)	00 00 00 00 00 00 01 00
	출력(I3)	30 02 F7 15 F9 EB DF 1F
	암호문(C3)	9E 2F 7D 42 E7 E8 73 83
	평문(03)	AE 2D 8A 57 1E 03 AC 9C
블록4	입력(P4)	00 00 00 00 00 00 01 01
	출력(I4)	E1 82 63 86 97 7D 7C 3F
	암호문(C4)	7F 35 0C 2A D2 D2 F2 6E
	평문(04)	9E B7 6F AC 45 AF 8E 51
블록5	입력(P5)	00 00 00 00 00 00 01 02
	출력(I5)	83 EA 79 8E 46 9D 83 f0
	암호문(C5)	B3 22 65 C8 E5 C1 67 E1
	평문(05)	30 C8 1C 46 A3 5C E4 11
블록6	입력(P6)	00 00 00 00 00 00 01 03
	출력(I6)	80 EB 00 4E 35 08 E6 f6
	암호문(C6)	65 10 C1 57 2F 02 B4 19
	평문(06)	E5 FB C1 19 1A 0A 52 EF
블록7	입력(P7)	00 00 00 00 00 00 01 04
	출력(I7)	F2 88 89 17 50 B9 7D 97
	암호문(C7)	04 17 AD 52 8F F6 E6 80
	평문(07)	F6 9F 24 45 DF 4F 9B 17
블록8	입력(P8)	00 00 00 00 00 00 01 05
	출력(I8)	8B 10 86 EA AD 0C 41 BB
	암호문(C8)	26 3B C7 91 4B 60 76 AB
	평문(08)	AD 2B 41 7B E6 6C 37 10

4. CMAC 모드 참조구현값

4.1 데이터 3

입력	키(Key)			F9 C5 9D D0 B2 8B B2 9B 74 1B C6 50 BE 41 86 BB
	평문(P)			BD C9 9B B2 7A E8 7C 9D 53 D7 86 35 A8 C1 4E 8B D6 DF 36 4C 69 E2 3E 44 8D 91 1D 22 C6 B3 91 7B E1 2F F3 11 2B 72 0E 54 E0 11 31 29 E6 AF
	MAC 길이			8 byte
처리 과정	보조 비밀키 K1, K2 생성		L	C3 63 B5 69 DA 88 97 07
			K1	86 C7 6A D3 B5 11 2E 15
			K2	0D 8E D5 A7 6A 22 5C 31
	MAC값 계산	2-0	m	6
			M_m	ED 9F E4 8E 8C 8D DC 31
		2-1	X	BD C9 9B B2 7A E8 7C 9D
			Y	5E BF 3B FD 43 D7 B1 DE
		2-2	X	0D 68 BD C8 EB 16 FF 55
			Y	34 2E 5D 20 A0 44 D1 C9
		2-3	X	E2 F1 6B 6C C9 A6 EF 8D
			Y	50 C4 FD 54 9D 31 DB B3
		2-4	X	DD 55 E0 76 5B 82 4A C8
			Y	FA 18 71 0F F4 92 B8 62
		2-5	X	1B 37 82 1E DF E0 B6 36
			Y	2E B8 F2 CA B1 A0 8A 47
		2-6	X	C3 27 16 44 3D 2D 56 76
			Y	17 26 86 65 76 B7 36 01
출력	MAC값(T)			17 26 86 65 76 B7 36 01

4.2 데이터 4

입력	키(Key)			50 59 62 0A 3A DF 41 59 C4 1B 3F 89 F4 8A 36 1E
	평문(P)			(NULL)
	MAC 길이			8 byte
처리 과정	보조 비밀키 K1, K2 생성		L	A1 C9 E9 98 82 50 7D 43
			K1	43 93 D3 31 04 A0 FA 9D
			K2	87 27 A6 62 09 41 F5 3A
	MAC값 계산	2-0	m	1
			M_m	07 27 A6 62 09 41 F5 3A
		2-1	X	07 27 A6 62 09 41 F5 3A
			Y	5B 79 67 90 48 15 24 D4
출력	MAC값(T)			5B 79 67 90 48 15 24 D4