SEED 블록암호 알고리즘에 대한 소스코드 활용 매뉴얼

2019. 10

KISA 한국인터넷진흥원

제·개정 이력

순번	제·개정일	제·개정 내역
1	2013.12	"SEED 블록암호 알고리즘에 대한 소스코드 활용 매뉴얼" 발간
2	2018.11	SEED 블록암호 운영모드(CCM, GCM) 추가
3	2019.10	SEED 블록암호 운영모드(CMAC) 추가 및 매뉴얼 개정

Contents

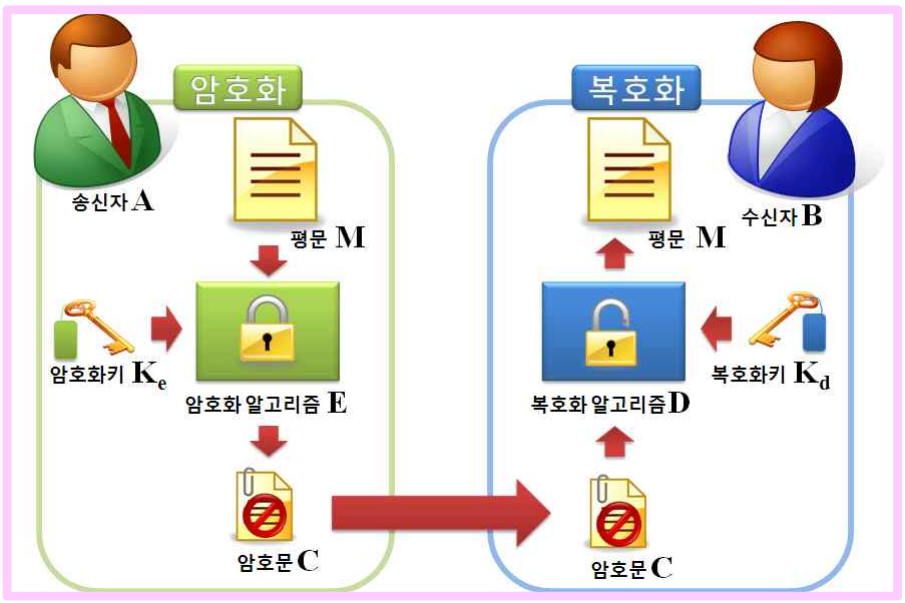
1. 개요

SEED는 전자상거래, 금융, 무선통신 등에서 전송되는 중요 정보를 보호하기 위해 1999년 2월 한국정보보호진흥원을 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.

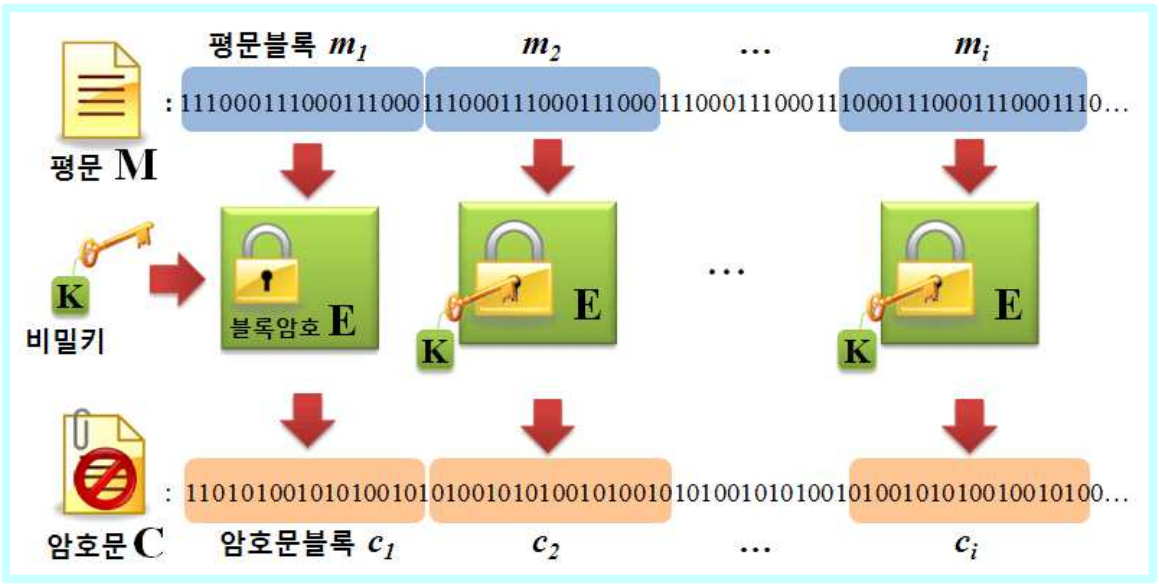
2. 블록암호 알고리즘

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.



< 암·복호화 과정 >

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

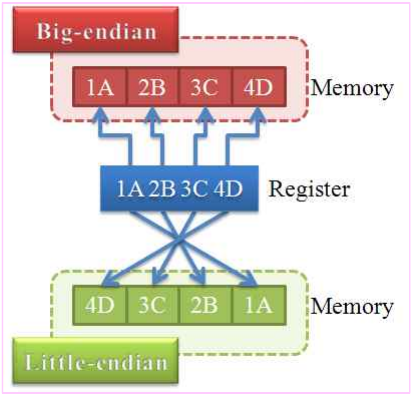
블록암호 SEED를 실제로 사용하기 위해서는 구현 환경에서 사용되는 엔디안(endianness)과 암호화된 데이터가 저장되는 데이터 형식(Data Type)을 고려해야 한다. 또한, 입력 블록을 블록암호에 적용하는 방법인 운영모드(Mode of operation)와 마지막 블록의 크기를 맞추기 위한 패딩(Padding)을 함께 구현해 주어야 한다.

가. 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 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 || __linux__
#define LITTLE_ENDIAN
#else
#define BIG_ENDIAN
#endif
```

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

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

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

나. 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.

다. 운영모드

운영모드란, 여러 개의 입력 블록들을 블록암호에 적용하여 암·복호화하는 방법에 대한 정의이다. 이러한 운영모드는 블록암호와는 독립적으로 정의된다. 대표적으로 가장 널리 이용되는 블록암호의 기밀성 운영모드에는 ECB(Electronic Code Book) 모드, CBC(Cipher Block Chaining) 모드, CFB(Ciphertext FeedBack) 모드, OFB(Output FeedBack) 모드, CTR(CounTeR) 모드, 인증 운영 모드에는 CMAC(Cipher-based MAC) 모드, 인증 암호화 운영모드에는 CCM(Counter with CBC-MAC) 모드, GCM(Galois/Counter Mode) 모드가 있다. 하지만 이번 매뉴얼에서는 KISA에서 개발하여 배포하는 ECB, CBC, CTR, CCM, GCM, CMAC에 대해서만 설명을 하도록 하겠다.

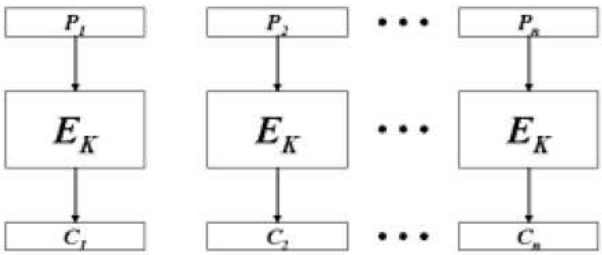
1) ECB 운영모드

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 : $P=\{P_1,\cdots,P_n\}$, 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 >

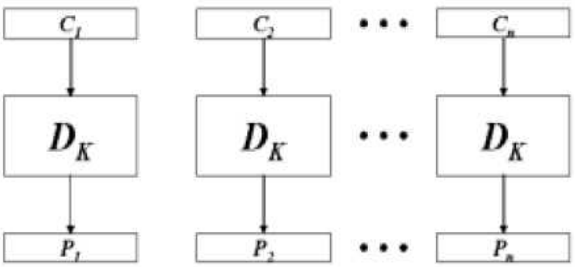


< ECB 암호화 >

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

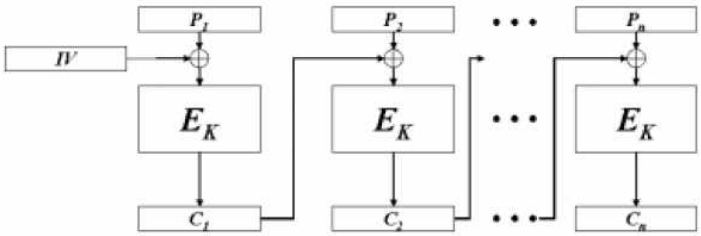
2) CBC 운영모드

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(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.

```
o 입력 : P={P1,···,Pn}, K, IV
o 출력 : C={C1,···,Cn}
o 처리과정
   C0 = IV
   for i=1 to n
        Ii = Pi ⊕ Ci-1
        Oi = EK(Ii)
        Ci = Oi
```

< CBC 암호화 의사코드 >

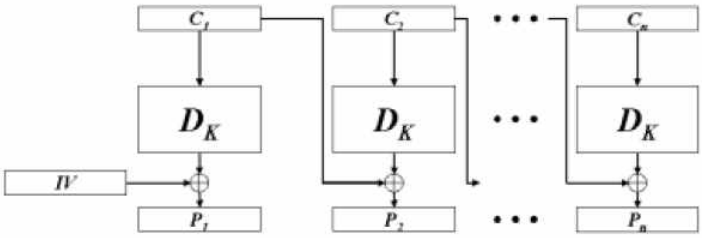


< CBC 암호화 >

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={C1,···,Cn}, K, IV
o 출력 : P={P1,···,Pn}
o 처리과정
   C0 = IV
   for i=1 to n
        Ii = Ci
        Oi = DK(Ii)
        Pi = Ci-1 ⊕ Oi
```

< CBC 복호화 의사코드 >



< CBC 복호화>

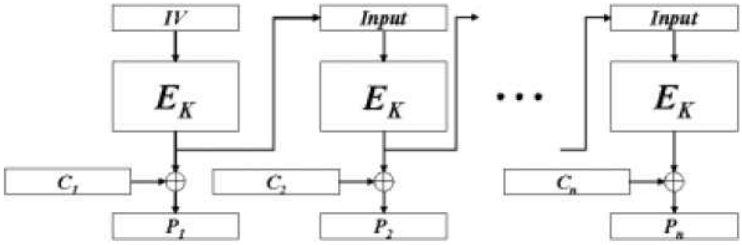
3) CTR 운영모드

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

< CTR 암호화 의사코드 >

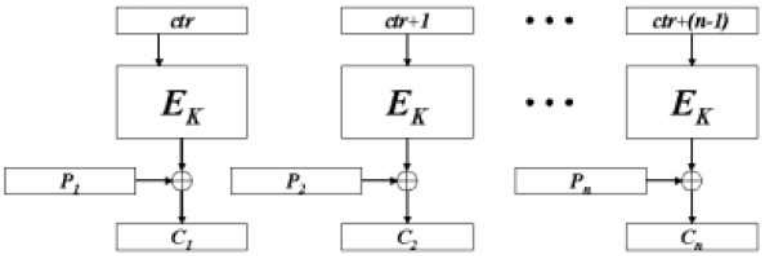


< CTR 암호화 >

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

< CTR 복호화 의사코드 >



< CTR 복호화 >

4) CCM 운영모드

CCM(Counter with CBC-MAC) 모드는 CTR 모드에 CBC-MAC의 메시지 인증을 포함시킨 128 비트 전용 운영모드이다.

CCM 모드의 암호화 과정은 입력 데이터(N, P, A)로부터 데이터 블록 열(B)과 초기 카운터 블록(CTR_0)을 생성하는 것으로 시작한다. 데이터 블록 열(B)은 CBC-MAC의 출력값을 얻는데 사용되며, 인증값(T)은 초기 카운터 블록(CTR_0)에 대한 암호화 함수의 출력 블록과 CBC-MAC 출력값의 XOR 결과를 최상위 비트부터 주어진 인증값 길이(Tlen)만큼 절삭한 값으로 설정한다. 그리고 $CTR_1 = CTR_0 + 1 \bmod 2^b$을 초기값(IV)으로 하여 CTR 모드를 통해 평문(P)에 대한 암호문(C)을 생성한다. 최종 출력은 CTR 모드로부터 얻어진 암호문(C)과 CBC-MAC으로부터 얻어진 인증값(T)을 연접한 결과로 그 길이는 (Plen+Tlen) 비트이다.

---

1 입력 : 난스 N (15-q 바이트),
부가 인증 데이터 A ($a < 2^{64}$),
평문 P ($Plen \le 2^{8q+3}-8$),
비밀 키 K

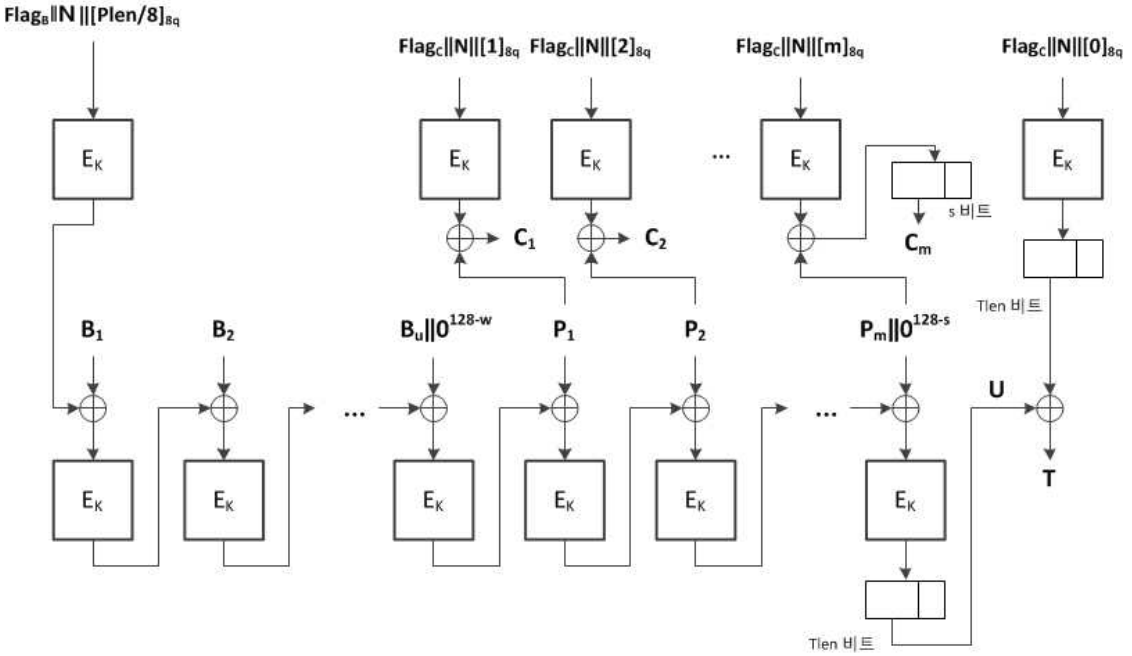
---

2 처리 과정
if Alen = 0 then
$Flag_B \leftarrow 0^2 \| [(t-2)/2]_3 \| [q-1]_3$;
else
$Flag_B \leftarrow 0 \| 1 \| [(t-2)/2]_3 \| [q-1]_3$;

endif
$B_0 \leftarrow Flag_B \parallel N \parallel [Plen/8]_{8q}$;
if $0 < a < 65280$ then
    $B \leftarrow B_0 \parallel [a]_{16} \parallel A$;
else if $65280 \le a < 2^{32}$ then
    $B \leftarrow B_0 \parallel 1^{15} \parallel 0 \parallel [a]_{32} \parallel A$;
else if $2^{32} \le a < 2^{64}$ then
    $B \leftarrow B_0 \parallel 1^{16} \parallel [a]_{64} \parallel A$;
endif
Partition B into $B_0 \parallel B_1 \parallel ... \parallel B_u$ such that
    $Len(B_0) = ... = Len(B_{u-1}) = b$ and $0 < w = Len(B_u) \le b$;
$B \leftarrow B \parallel 0^{b-w} \parallel P$;
Partition B into $B_0 \parallel B_1 \parallel ... \parallel B_r$ such that
    $Len(B_0) = ... = Len(B_{r-1}) = b$ and $0 < s = Len(B_r) \le b$;
$B \leftarrow B \parallel 0^{b-s}$;
$Y \leftarrow E_K(B_0)$;
for i from 1 to r do
    $X \leftarrow B_i \oplus Y$;
    $Y \leftarrow E_K(X)$;
endfor
$T \leftarrow MSB_{Tlen}(Y)$;
$Flag_C \leftarrow 0^5 \parallel [q-1]_3$;
$CTR_0 \leftarrow Flag_C \parallel N \parallel 0^{8q}$;
$S_0 \leftarrow E_K(CTR_0)$;
$T \leftarrow T \oplus MSB_{Tlen}(S_0)$;
for i from 1 to (m-1) do
    $S_i \leftarrow E_K(CTR_i)$;
    $C_i \leftarrow P_i \oplus S_i$;
endfor
$S_m \leftarrow E_K(CTR_m)$;
$C_m \leftarrow P_m \oplus MSB_{Len(Pm)}(S_m)$;

3 출력 : 암호문 $C = C_1 \parallel C_2 \parallel ... \parallel C_m$, 인증값 T

< CCM 암호화 의사코드 >



< CCM 암호화 >

CCM 모드의 복호화 과정은 입력 데이터(N, (C, T), A)로부터 초기 카운터 블록(CTR_0)을 생성하는 것으로 시작한다. 초기 카운터 블록(CTR_0)에 대한 암호화 함수 출력 블록은 입력 데이터에 포함된 인증값(T)과 XOR하며, 그 결과는 마지막 단계에서 CBC-MAC의 출력값과 비교하는 데 사용된다. 이어서 암호문(C)에 대한 평문(P)은 $CTR_1 = CTR_0 + 1 \bmod 2^b$을 초기값(IV)으로 한 CTR 모드를 통해 생성된다. 이어서 데이터 블록 열(B)을 생성한 후, 이에 대한 CBC-MAC의 출력값을 최상위 비트부터 주어진 인증값 길이(Tlen)만큼 절삭하여 인증값(T_1)으로 설정한다. 마지막으로 복호화 과정에서 계산된 인증값(T_1)과 입력 데이터에 포함된 인증값(T)을 비교하여, 일치할 경우 평문(P)을 출력하고 그렇지 않을 경우 오류 플래그를 출력한다.

---

1 입력 : 난스 N (임의 길이의 비트열),
부가 인증 데이터 A (임의 길이의 비트열),
암호문 C (임의 길이의 비트열),
인증값 T (임의 길이의 비트열),
비밀 키 K

---

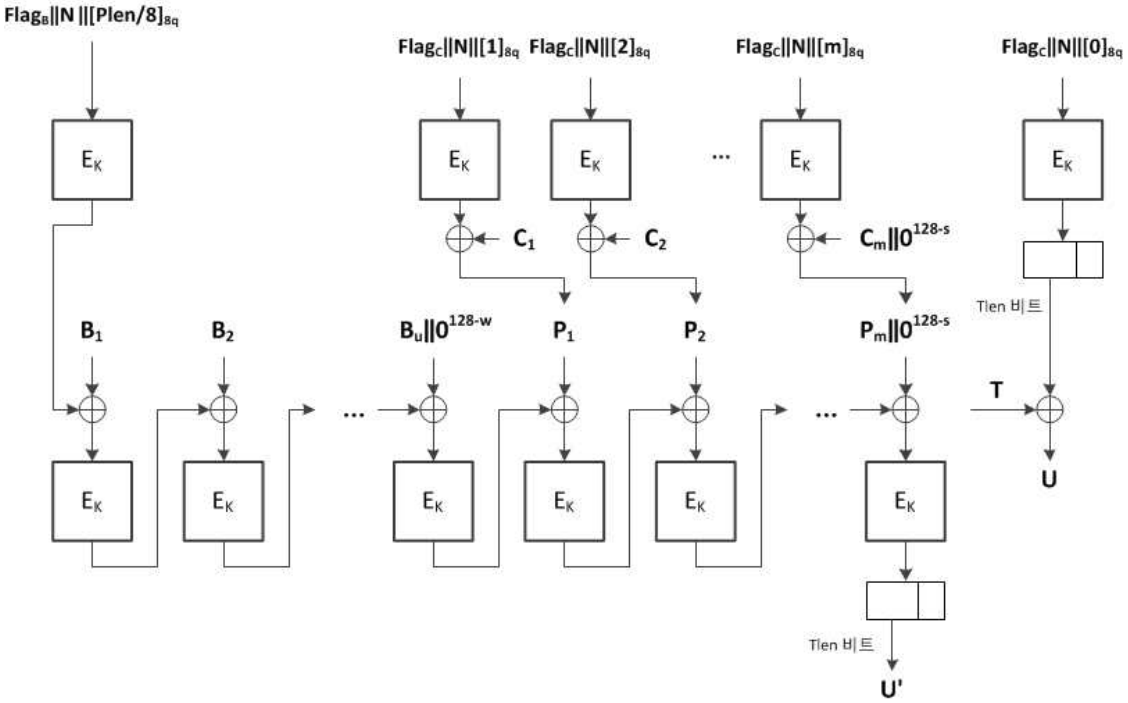
2 처리 과정

$Flag_C \leftarrow 0^5 \| [q-1]_3$;
$CTR_0 \leftarrow Flag_C \| N \| 0^{8q}$;
$S_0 \leftarrow E_K(CTR_0)$;
$T \leftarrow T \oplus MSB_{Tlen}(S_0)$;
if Clen > 0 then
  Partition C into $C_1 \| \ldots \| C_m$ such that
    $Len(C_1) = \ldots = Len(C_{m-1}) = b$ and $0 < Len(C_m) \leq b$;
  for i from 1 to (m-1) do
    $S_i \leftarrow E_K(CTR_i)$;

$C_i \leftarrow P_i \oplus S_i$;
endfor
$S_m \leftarrow E_K(CTR_m)$;
$P_m \leftarrow C_m \oplus MSB_{Len(Cm)}(S_m)$;
endif
if N, A or P is not valid(see Section 7.1.2.), then
halt and output INVALID;
endif
if Alen = 0 then
$Flag_B \leftarrow 0^2 \| [(t - 2)/2]_3 \| [q - 1]_3$;
else
$Flag_B \leftarrow 0 \| 1 \| [(t - 2)/2]_3 \| [q - 1]_3$;
endif
$B_0 \leftarrow Flag_B \| N \| [Plen/8]_{8q}$;
if $0 < a < 65280$ then
$B \leftarrow B_0 \| [a]_{16} \| A$;
else if $65280 \leq a < 2^{32}$ then
$B \leftarrow B_0 \| 1^{15} \| 0 \| [a]_{32} \| A$;
else if $2^{32} \leq a < 2^{64}$ then
$B \leftarrow B_0 \| 1^{16} \| [a]_{64} \| A$;
endif
Partition B into $B_0 \| B_1 \| ... \| B_u$ such that
$Len(B_0) = ... = Len(B_{u-1}) = b$ and $0 < w = Len(B_u) \leq b$;
$B \leftarrow B \| 0^{b-w} \| P$;
Partition B into $B_0 \| B_1 \| ... \| B_r$ such that
$Len(B_0) = ... = Len(B_{r-1}) = b$ and $0 < s = Len(B_r) \leq b$;
$B \leftarrow B \| 0^{b-s}$;
$Y \leftarrow E_K(B_0)$;
for i from 1 to r do
$X \leftarrow B_i \oplus Y$;
$Y \leftarrow E_K(X)$;
endfor
$T_1 \leftarrow MSB_{Tlen}(Y)$;
if $T \neq T_1$ then
return INVALID;
else
return $P = P_1 \| ... \| P_m$;
endif

3 출력 : 평문 P 또는 INVALID

< CCM 복호화 의사코드 >



< CCM 복호화 >

5) GCM 운영모드

GCM(Galois/Counter Mode) 모드는 CTR 모드에 유한체 GF(2^{128}) 상의 곱 연산을 이용한 메시지 인증을 포함시킨 128 비트 전용 운영모드이다.

GCM의 암호화 과정은 먼저 인증값 계산에 사용할 보조 비밀 키(H)와 초기 카운터 블록(CTR_0)을 생성하는 것으로 시작한다. 그리고 하위 32 비트에 대해 1 증가시킨 카운터 블록(CTR_1)을 초기값(IV)으로 하여 CTR 모드를 통해 평문(P)에 대한 암호문(C)을 생성한다. 암호문(C)과 부가 인증 데이터(A)는 GHASH 함수의 출력값을 얻는데 사용되며, 인증값(T)은 초기 카운터(CTR_0)에 대한 암호화 함수의 출력 블록과 GHASH 함수 출력의 XOR 결과를 최상위 비트부터 주어진 인증값 길이(Tlen) 만큼 절삭한 값으로 설정한다. 최종 출력은 CTR 모드로부터 얻어진 암호문(C)과 GHASH 함수로부터 얻어진 인증값(T)을 연접한 결과로 그 길이는 (Plen+Tlen) 비트이다.

---

1 입력 : 난스 N (0 < Len(N) < 2^{64}),
부가 인증 데이터 A (Alen < 2^{64}), 평문 P (Plen ≤ 2^{39} - 256),
비밀 키 K

---

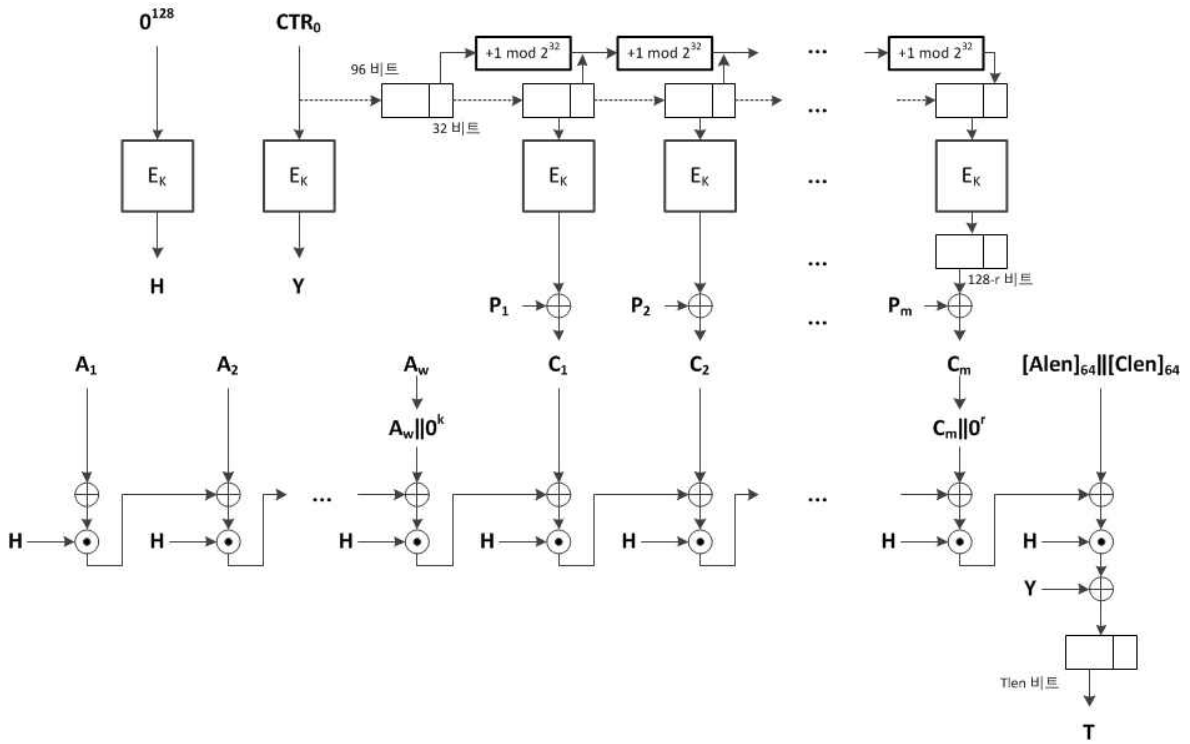
2 처리 과정

$H \leftarrow E_K(0^{128})$;
if Len(N) = 96 then
$CTR_0 \leftarrow N \| 0^{31} \| 1$;
else
$s \leftarrow 128 \lceil Len(N)/128 \rceil - Len(N)$;
$CTR_0 \leftarrow GHASH(H, N \| 0^{s+64} \| [Len(N)]_{64})$;
endif

r ← 128 ⌈ Plen/128 ⌉ - Plen;
if Plen > 0 then
Partition P into P_1 ‖ ... ‖ P_m such that
Len(P_1) = ... = Len(P_{m-1}) = 128 and 0 < r = Len(P_m) ≤ 128;
for i from 1 to (m-1) do
CTR_i ← $MSB_{96}(CTR_{i-1})$ ‖ [BitToInt($LSB_{32}(CTR_{i-1})$) + 1 mod $2^{32}]_{32}$;
C_i ← P_i ⊕ $E_K(CTR_i)$;
endfor
CTR_m ← $MSB_{96}(CTR_{m-1})$ ‖ [BitToInt($LSB_{32}(CTR_{m-1})$) + 1 mod $2^{32}]_{32}$;
C_m ← P_m ⊕ $MSB_{128-r}(E_K(CTR_m))$;
endif
k ← 128 ⌈ Alen/128 ⌉ - Alen;
S ← GHASH(H, A ‖ 0^k ‖ C ‖ 0^r ‖ $[Alen]_{64}$ ‖ $[Clen]_{64}$);
Y ← $E_K(CTR_0)$;
T ← MSB_{Tlen}(Y ⊕ S);

3 출력 : 암호문 C, 인증값 T

< GCM 암호화 의사코드 >



< GCM 암호화 >

GCM의 복호화 과정은 먼저 인증값 계산에 사용할 보조 비밀 키(H)와 초기 카운터 블록(CTR_0)을 생성하는 것으로 시작한다. 그리고 하위 32 비트에 대해 1 증가시킨 카운터 블록(CTR_1)을 초기값(IV)으로 한 CTR 모드를 통해 암호문(C)에 대한 평문(P)을 생성한다. 이이서 인증값(T_1)은 초기 카운터(CTR_0)에 대한 암호화 함수의 출력 블록과 GHASH 함수 출력의 XOR 결과를 최상위 비트부터 주어진 인증값 길이(Tlen)만큼 절삭한 값으로 설정한다. 마지막으로 복호화 과정에서 계산된 인증값(T_1)과 입력 데이터에 포함된 인증값(T)을 비교하여, 일치할 경우 평문(P)을 출력하고 그렇지 않

을 경우 오류 플래그를 출력한다.

---

1 입력 : 난스 N (임의 길이의 비트열),
부가 인증 데이터 A (임의 길이의 비트열),
암호문 C (임의 길이의 비트열),
인증값 T (임의 길이의 비트열),
비밀 키 K

---

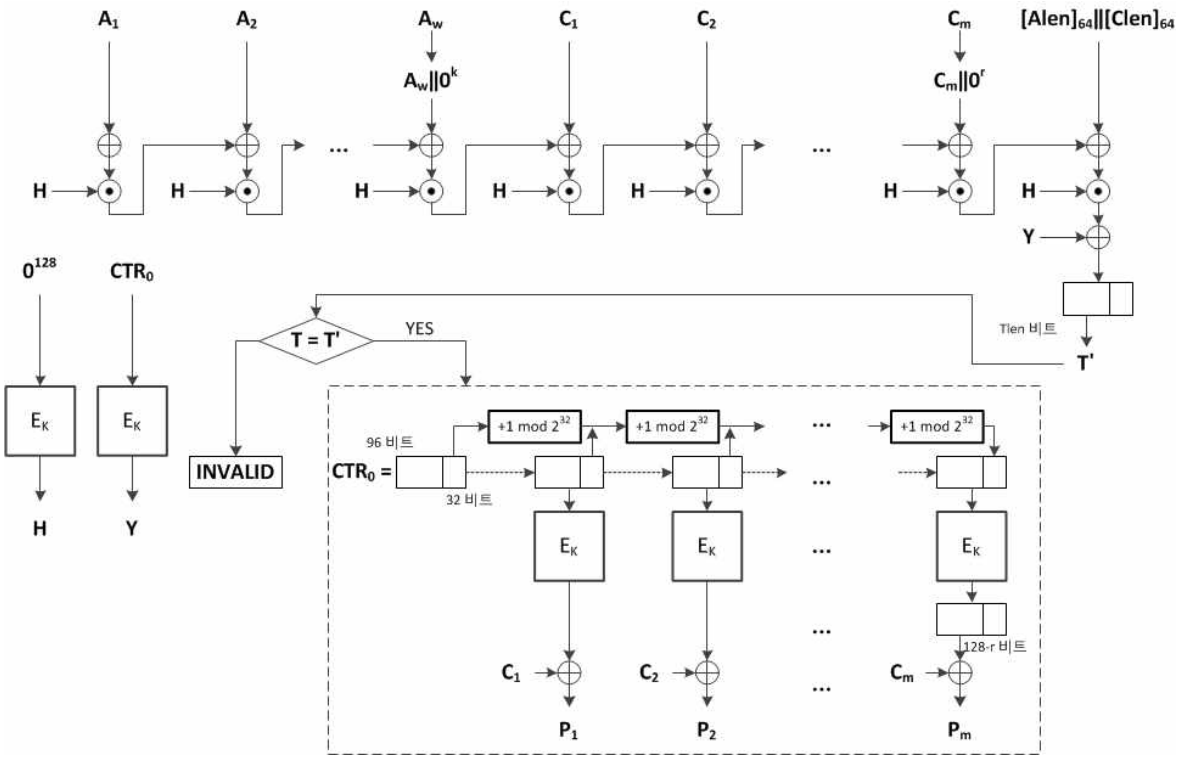
2 처리 과정

if Len(N), Len(A) or Len(C) are not supported then
    halt and output INVALID;
endif
if Len(T) ≠ Tlen then
    halt and output INVALID;
endif
$H \leftarrow E_K(0^{128})$;
if Len(N) = 96 then
    $CTR_0 \leftarrow N \| 0^{31} \| 1$;
else
    $s \leftarrow 128 \lceil Len(N)/128 \rceil - Len(N)$;
    $CTR_0 \leftarrow GHASH(H, N \| 0^{s+64} \| [Len(N)]_{64})$;
endif
$r \leftarrow 128 \lceil Clen/128 \rceil - Clen$;
if Clen > 0 then
    Partition C into $C_1 \| \ldots \| C_m$ such that
        $Len(C_1) = \ldots = Len(C_{m-1}) = 128$ and $0 < Len(C_m) \leq 128$;
    for i from 1 to (m-1) do
        $CTR_i \leftarrow MSB_{96}(CTR_{i-1}) \| [BitToInt(LSB_{32}(CTR_{i-1})) + 1 \bmod 2^{32}]_{32}$;
        $P_i \leftarrow C_i \oplus E_K(CTR_i)$;
    endfor
    $CTR_m \leftarrow MSB_{96}(CTR_{m-1}) \| [BitToInt(LSB_{32}(CTR_{m-1})) + 1 \bmod 2^{32}]_{32}$;
    $P_m \leftarrow C_m \oplus MSB_{128-r}(E_K(CTR_m))$;
endif
$k \leftarrow 128 \lceil Alen/128 \rceil - Alen$;
$S \leftarrow GHASH(H, A \| 0^k \| C \| 0^r \| [Alen]_{64} \| [Clen]_{64})$;
$Y \leftarrow E_K(CTR_0)$;
$T_1 \leftarrow MSB_{Tlen}(Y \oplus S)$;
if $T_1 \neq T$ then
    output INVALID;
else
    output P;
endif

---

3 출력 : 평문 P 또는 INVALID

< GCM 복호화 의사코드 >



< GCM 복호화 >

6) CMAC 운영모드

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

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.

1 입력 : 메시지 $M = M_1 \| M_2 \| \dots \| M_m^*$ ($Len(M_i) = b$ ($1 \le i \le (m-1)$), $Len(M_m^*) \le b$),
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
    $K1 \leftarrow L \ll 1$;
else
    $K1 \leftarrow (L \ll 1) \oplus R_b$;
endif
if $MSB_1(K1) = 0$ then
    $K2 \leftarrow K1 \ll 1$;
else
    $K2 \leftarrow (K1 \ll 1) \oplus R_b$;
endif

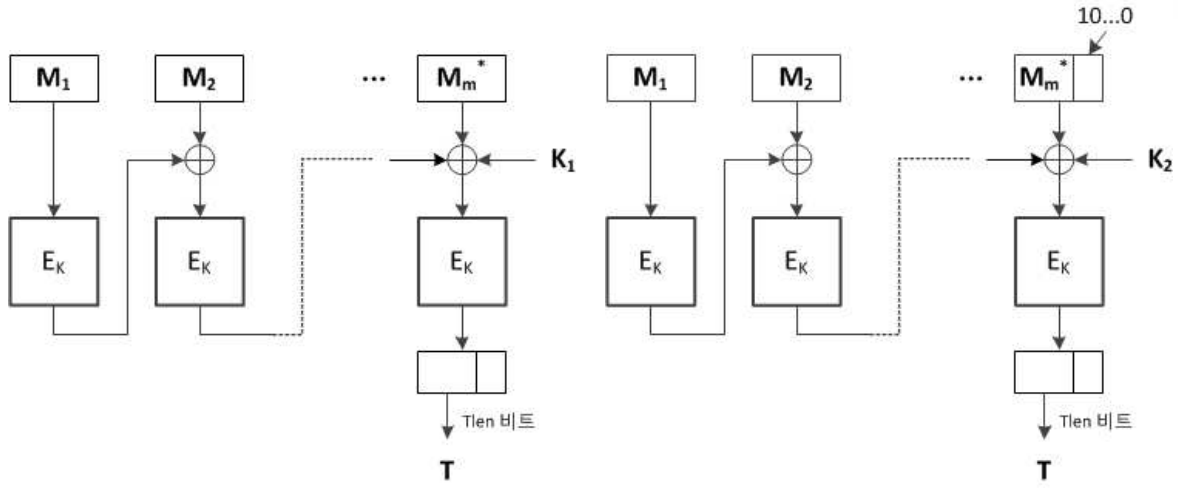
(2) 인증값 계산:

if Mlen = 0 then
    $m \leftarrow 1$;
else
    $m \leftarrow \lceil Mlen/b \rceil$;
endif

if $Len(M_m^*) = b$ then
    $M_m \leftarrow K1 \oplus M_m^*$;
else
    $M_m \leftarrow K2 \oplus (M_m^* \parallel 10^j)$, where $j = b - Len(M_m^*) - 1$;
endif
$Y \leftarrow 0^b$;
for i from 1 to m do
    $X \leftarrow Y \oplus M_i$;
    $Y \leftarrow E_K(X)$;
endfor
$T \leftarrow MSB_{Tlen}(Y)$;

3 출력 : 인증값 T

< CMAC 인증값 생성 의사코드 >



Len(M_m^*) = b일 때 Len(M_m^*) ≠ b일 때

< 보조 비밀 키를 사용한 CMAC 인증값 생성 >

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

라. 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.

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(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
```

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(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) 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
```

마. 활용 방법

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 블

SEED 알고리즘은 16Byte의 비밀키, 16Byte의 평문을 입력으로 16Byte의 암호문을 출력하는 블록암호 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 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

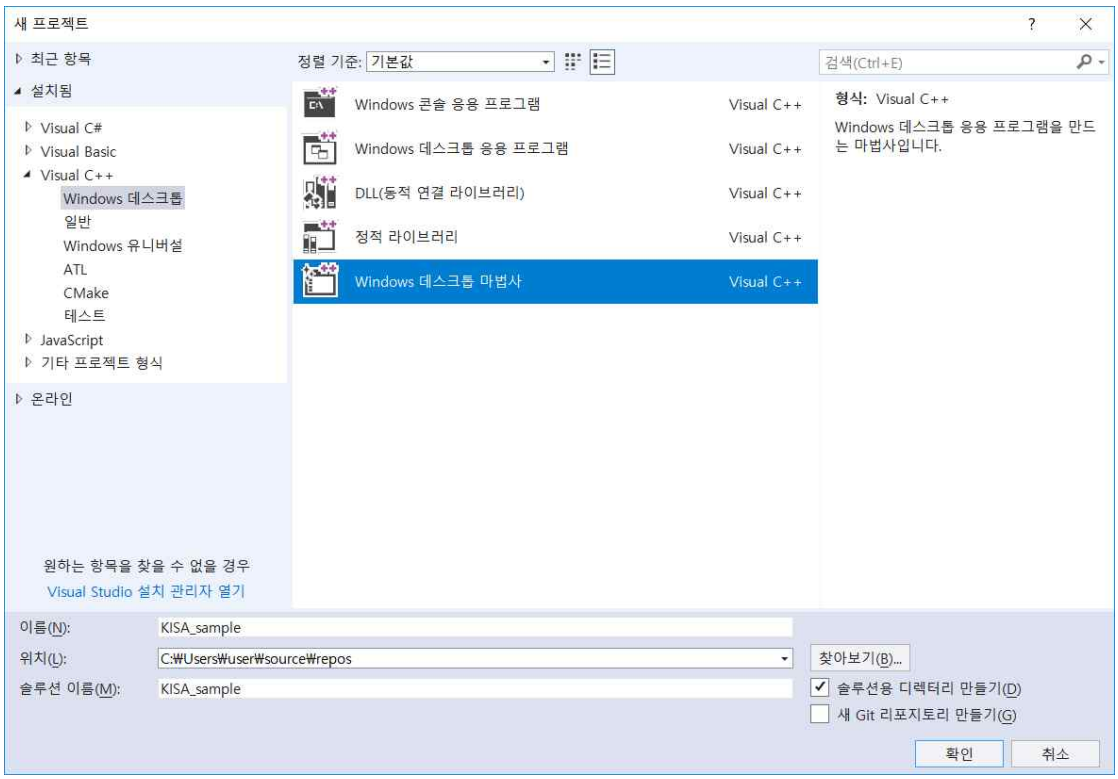
Windows, Linux 등 다양한 운영체제 환경에서 응용 프로그램을 개발과 함께 암호화를 적용할 수 있도록 기본적으로 많이 사용하는 C/C++ 및 Java를 기반으로 EC-KCDSA 전자서명 암호알고리즘 소스코드를 개발하였다.

가. C/C++

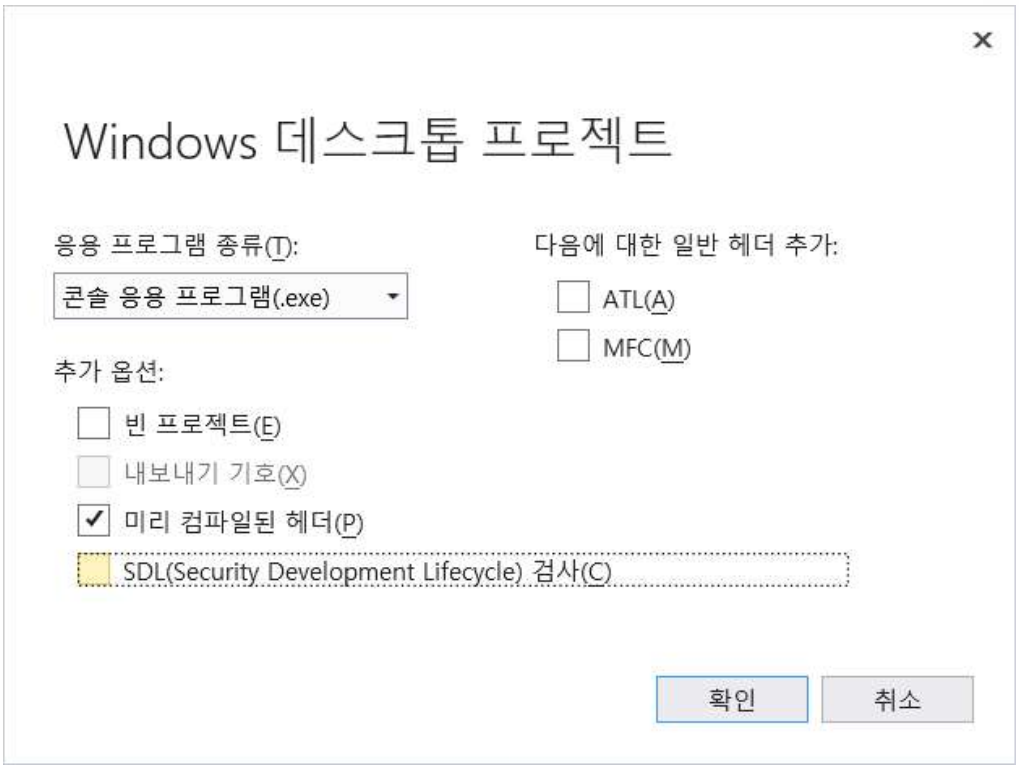
프로젝트 생성 및 소스 추가, 빌드 등 배포되는 소스코드를 이용하여 암호화/복호화를 실행하는 방법에 대해서는 Microsoft 社의 Visual studio 2017을 활용하여 설명하도록 한다.

1) 프로젝트 생성 및 빌드

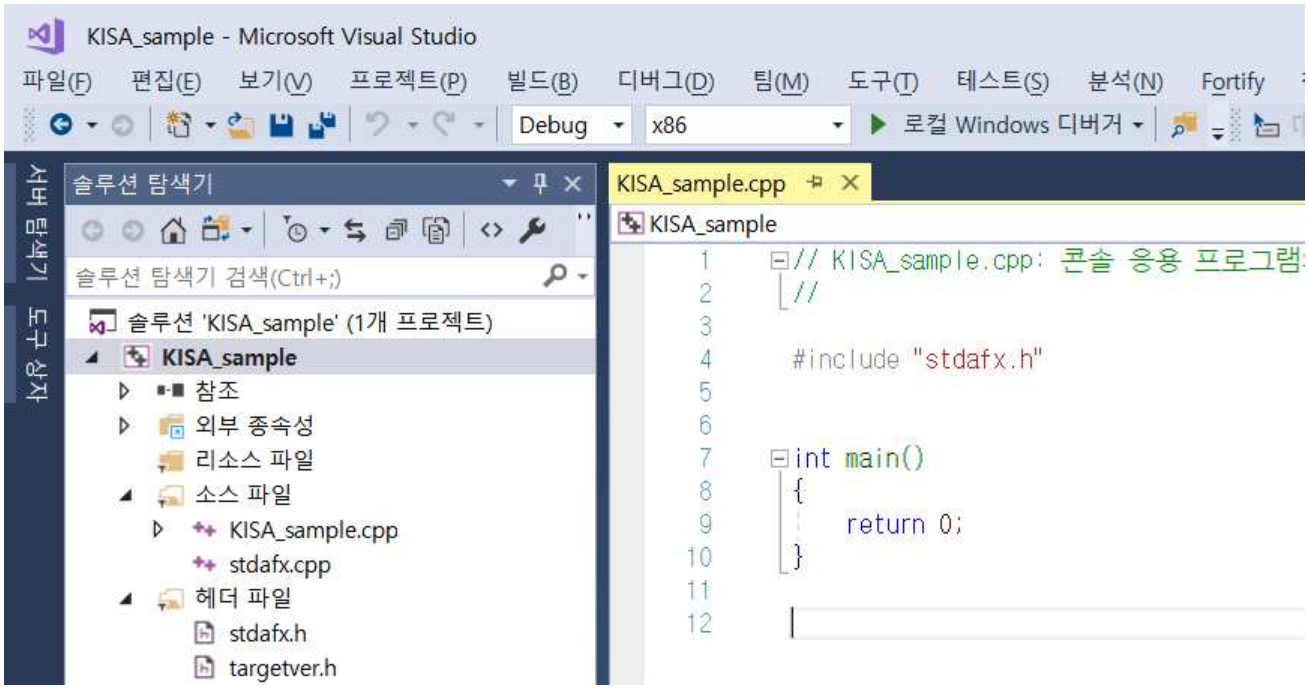
Visual studio를 실행하여 "Windows 데스크톱 마법사"를 선택한다. 이름에는 프로젝트 명을 기입하고 위치에는 생성시키고자 하는 곳의 위치를 지정하여 준다.



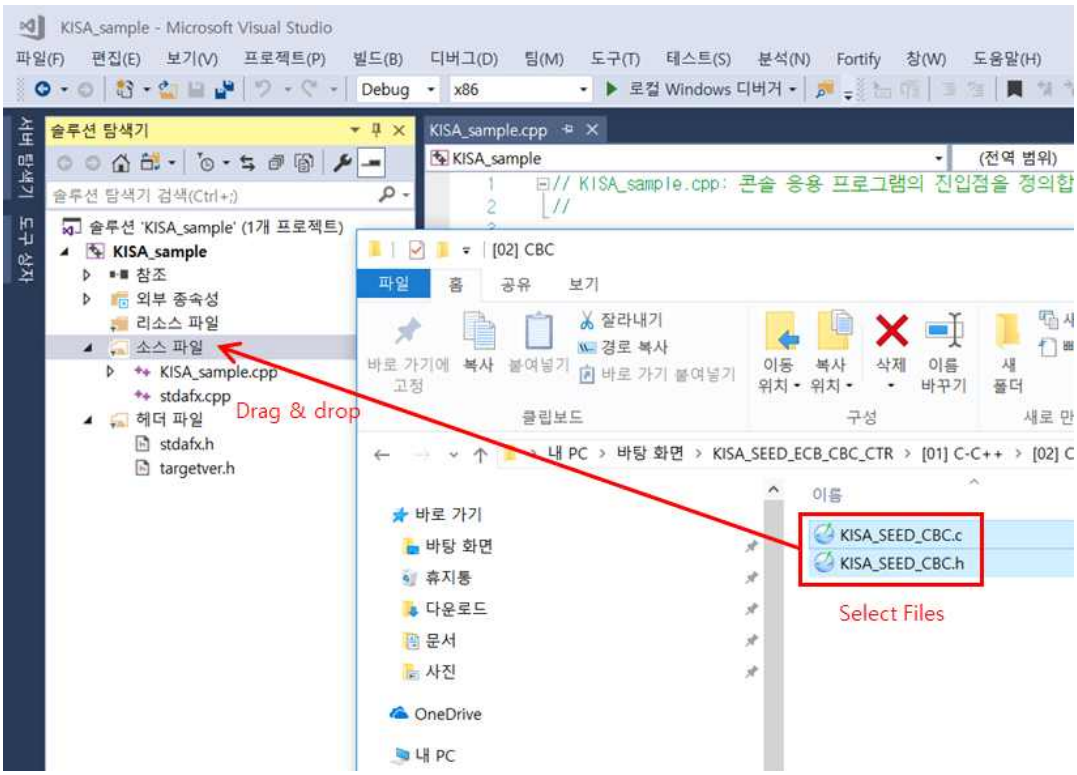
"콘솔 응용 프로그램"으로 선택한 후 마침 버튼을 누르면 콘솔 형태의 프로젝트가 생성된다. 미리 컴파일된 헤더를 체크해주면 int _tmain(int argc, TCHAR *argv[])을 자동으로 생성 시켜 준다. 빈 프로젝트는 말그대로 아무것도 없는 것으로 직접 헤더 파일과 소스파일을 추가하여 전부 작성하는 것이다. 여기서는 미리 컴파일된 헤더를 사용하여 프로젝트를 구성하도록 한다.



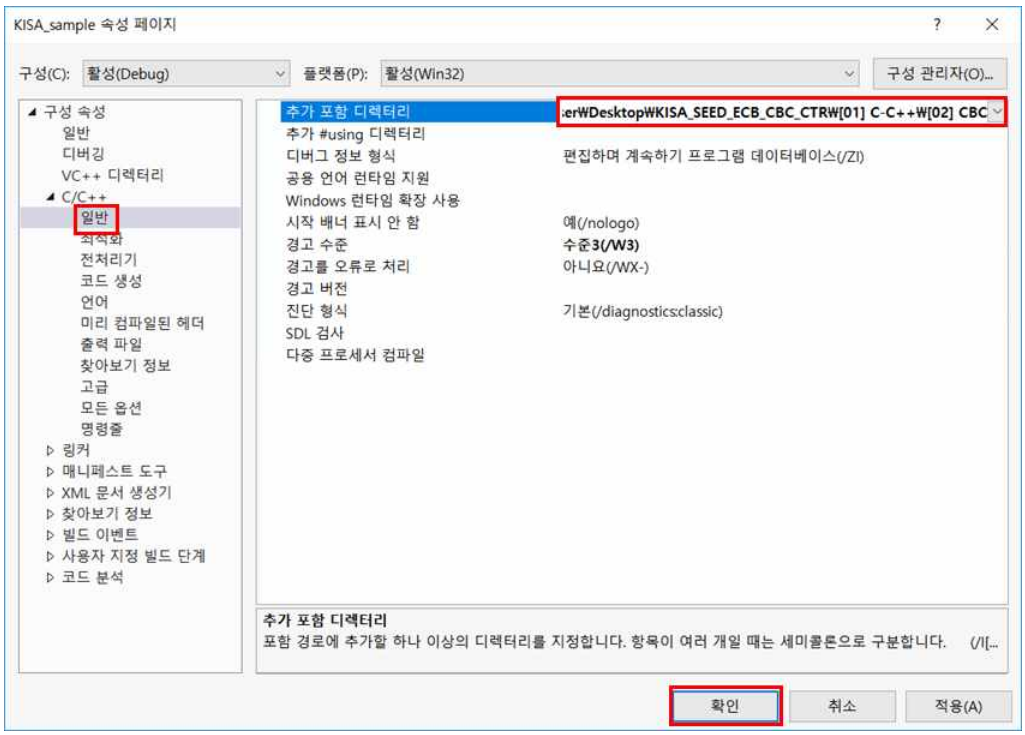
프로젝트 생성이 완료되면 "콘솔 응용 프로그램"을 작성할 수 있도록 기본적인 .h와 .cpp의 파일들을 생성된다. KISA_sample.cpp의 main 함수 내에서 암호화/복호화 소스를 활용하도록 한다.



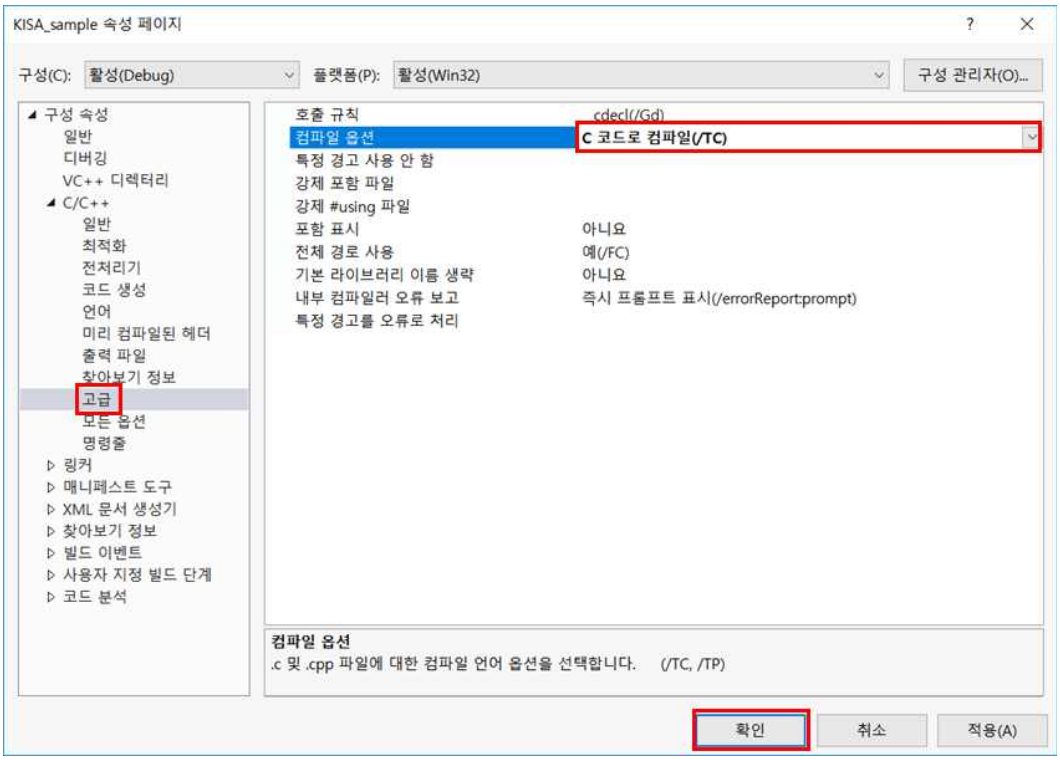
프로젝트 생성이 완료된 후, KISA에서 배포하는 소스코드 "KISA_SEED_xxx.c" 및 "KISA_SEED_xxx.h" 파일을 드래그 앤 드롭하여 소스 파일 폴더로 복사한다.



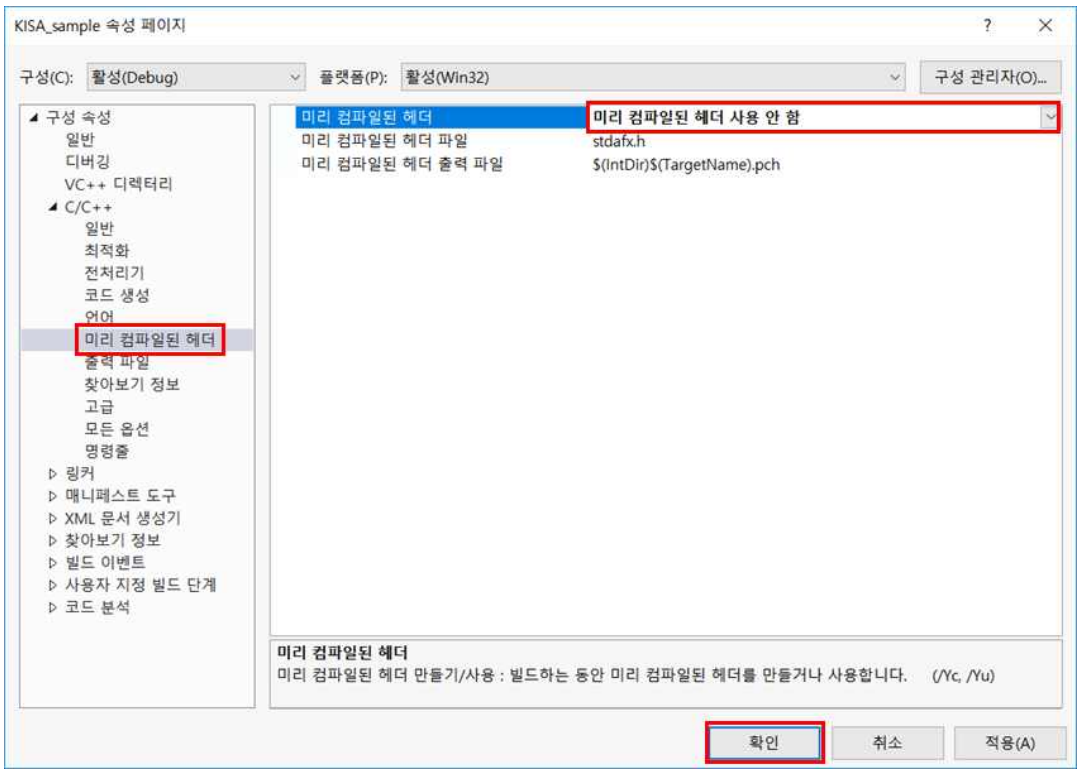
프로젝트 속성에서 C/C++의 추가 포함 디렉터리 항목에 포함 시킨 헤더와 소스 파일이 있는 폴더의 절대 경로 또는 상대 경로를 포함시켜야 한다. 여러 개의 폴더 또한 포함 시킬 수 있으며 구분자는 ";"로 하여 연속하여 기입 하면 된다.



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.



2) 소스코드 설명

가) SEED-ECB

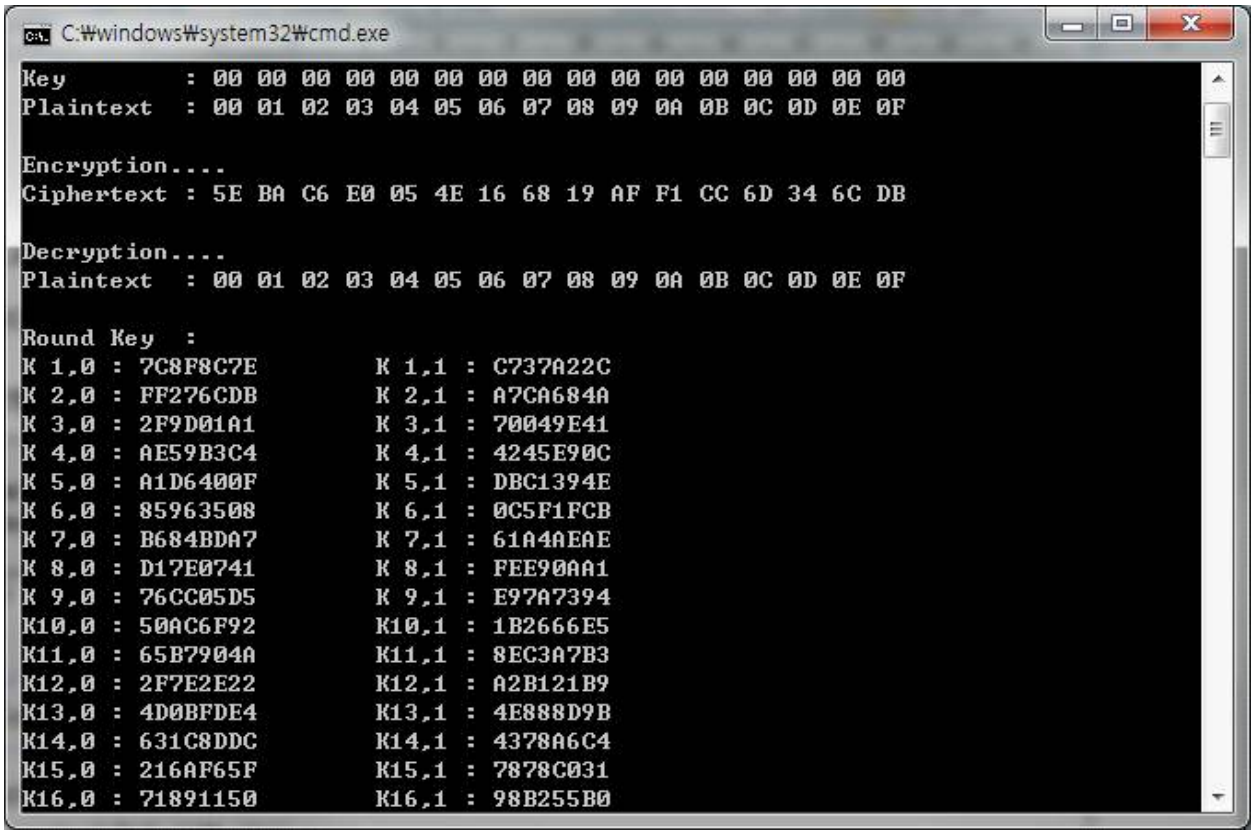
① 함수 설명

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.

② 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

나) SEED-CBC

① 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 블럭으로 처리함으로 pbszCipherText는 평문보다 16바이트 커야 한다. (평문이 16바이트 블럭 시 패딩 데이타가 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 블럭 암호화/복호화 함수

매개변수 :

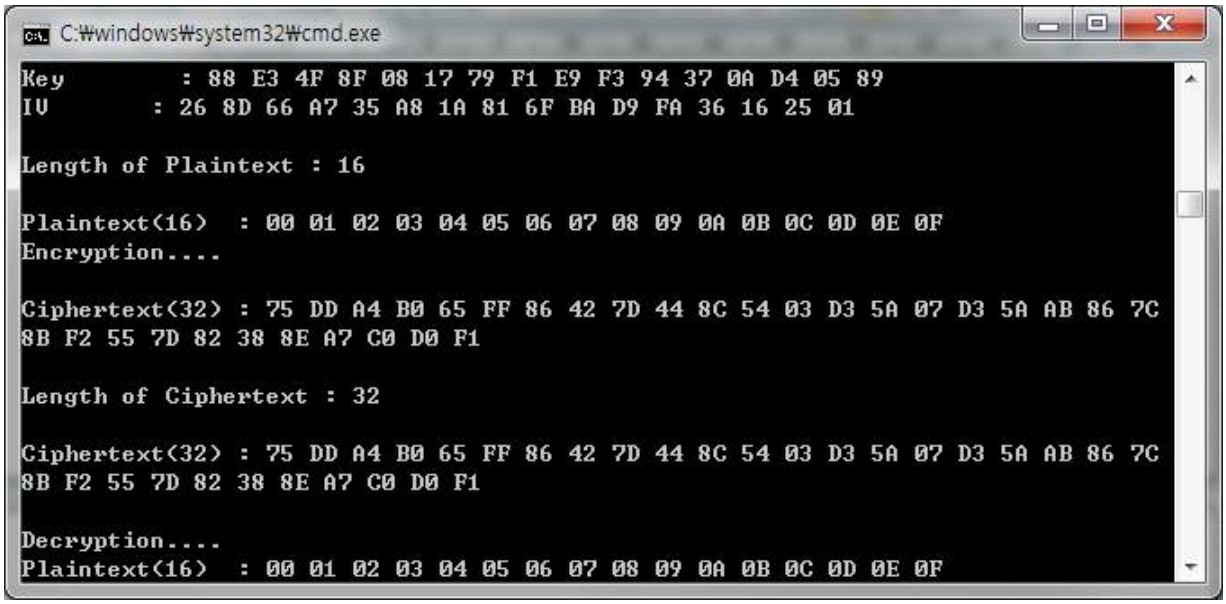
pInfo	SEED_CBC_init에서 설정된 KISA_HIGHT_INFO 구조체
in	평문/암호문 (평문은 chartoint32_for_SEED_CBC를 사용하여 int로 변환된 데이터)
inLen	평문/암호문 길이(BYTE 단위)
out	평문/암호문 버퍼
outLen	진행된 평문/암호문의 길이(BYTE 단위로 넘겨준다)

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_Process를 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.)

② 테스트 페이지



다) SEED-CTR

① 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 블럭으로 처리함으로 pbszCipherText는 평문보다 16바이트 커야 한다. (평문이 16바이트 블럭 시 패딩 데이타가 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 (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_CTR_Process(OUT KISA_SEED_INFO *pInfo, IN DWORD *in, IN int inLen, OUT DWORD *out, OUT int *outLen)

SEED CTR 다중 블럭 암호화/복호화 함수

매개변수 :

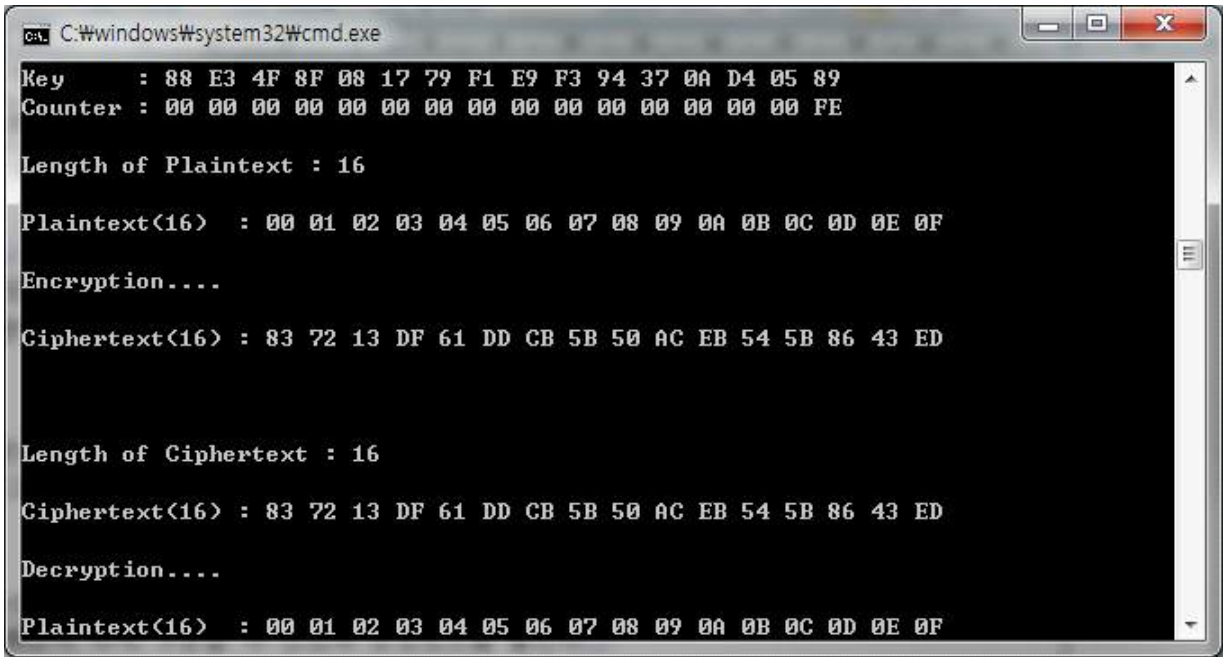
pInfo	SEED_CTR_init에서 설정된 KISA_HIGHT_INFO 구조체
in	평문/암호문 (평문은 chartoint32_for_SEED_CTR를 사용하여 int로 변환된 데이터)
inLen	평문/암호문 길이(BYTE 단위)
out	평문/암호문 버퍼
outLen	진행된 평문/암호문의 길이(BYTE 단위로 넘겨준다)

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

② 테스트 페이지



라) SEED-CCM

① 함수 설명

int SEED_CCM_Encryption(unsigned char *ct, unsigned int *ctLen, unsigned char *pt, unsigned int ptLen, unsigned int macLen, unsigned char *nonce, unsigned int nonceLen, unsigned char *aad, unsigned int aadLen, unsigned char *mKey)

SEED-CCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
ctLen	암호문과 MAC의 길이
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 생성 및 암호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

int SEED_CCM_Decryption(unsigned char *pt, unsigned int *ptLen, unsigned char *ct, unsigned int ctLen, unsigned int macLen, unsigned char *nonce, unsigned int nonceLen, unsigned char *aad, unsigned int aadLen, unsigned char *mKey)

SEED-CCM 알고리즘 MAC 검증 및 복호화 함수

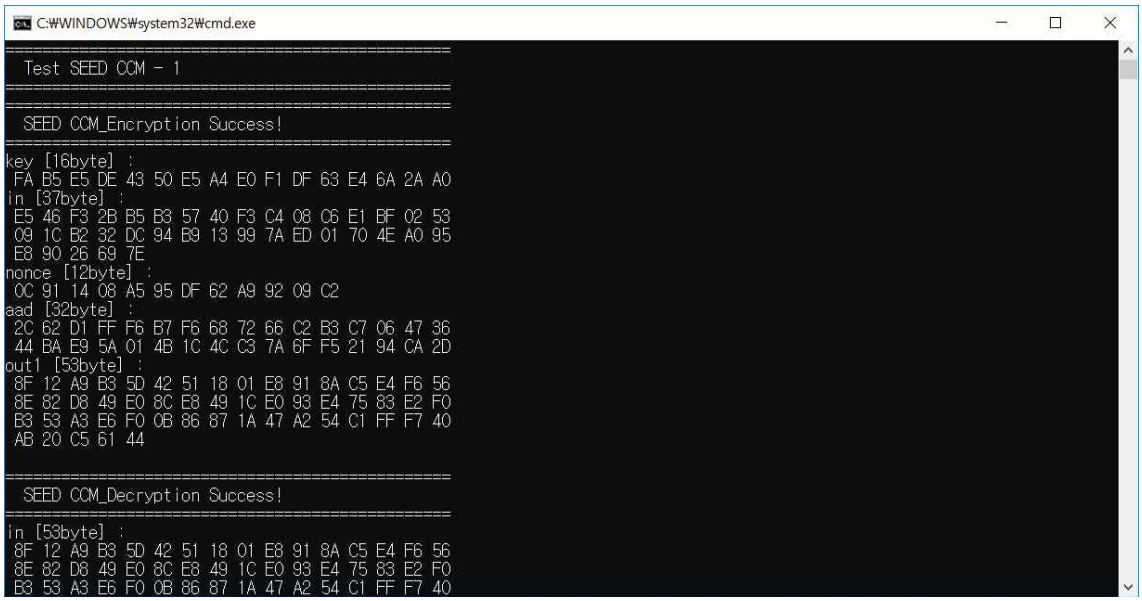
매개변수 :

pt	복호문 출력 버퍼
ptLen	복호문 길이
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 검증 및 복호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지


C:₩WINDOWS₩system32₩cmd.exe
Test SEED CCM - 1
SEED CCM_Encryption Success!
key [16byte] :
FA B5 E5 DE 43 50 E5 A4 E0 F1 DF 63 E4 6A 2A A0
in [37byte] :
E5 46 F3 2B B5 B3 57 40 F3 C4 08 C6 E1 BF 02 53
09 1C B2 32 DC 94 B9 13 99 7A ED 01 70 4E A0 95
E8 90 26 69 7E
nonce [12byte] :
0C 91 14 08 A5 95 DF 62 A9 92 09 C2
aad [32byte] :
2C 62 D1 FF F6 B7 F6 68 72 66 C2 B3 C7 06 47 36
44 BA E9 5A 01 4B 1C 4C C3 7A 6F F5 21 94 CA 2D
out1 [53byte] :
8F 12 A9 B3 5D 42 51 18 01 E8 91 8A C5 E4 F6 56
8E 82 D8 49 E0 8C E8 49 1C E0 93 E4 75 83 E2 F0
B3 53 A3 E6 F0 0B 86 87 1A 47 A2 54 C1 FF F7 40
AB 20 C5 61 44
SEED CCM_Decryption Success!
in [53byte] :
8F 12 A9 B3 5D 42 51 18 01 E8 91 8A C5 E4 F6 56
8E 82 D8 49 E0 8C E8 49 1C E0 93 E4 75 83 E2 F0
B3 53 A3 E6 F0 0B 86 87 1A 47 A2 54 C1 FF F7 40

마) SEED-GCM

① 함수 설명

int SEED_GCM_Encryption(unsigned char *ct, unsigned int *ctLen, unsigned char *pt, unsigned int ptLen, unsigned int macLen, unsigned char *nonce, unsigned int nonceLen, unsigned char *aad, unsigned int aadLen, unsigned char *mKey)

SEED-GCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
ctLen	암호문과 MAC의 길이
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :
- 0 : MAC 생성 및 암호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

int SEED_GCM_Decryption(unsigned char *pt, unsigned int *ptLen, unsigned char *ct, unsigned int ctLen, unsigned int macLen, unsigned char *nonce, unsigned int nonceLen, unsigned char *aad, unsigned int aadLen, unsigned char *mKey)

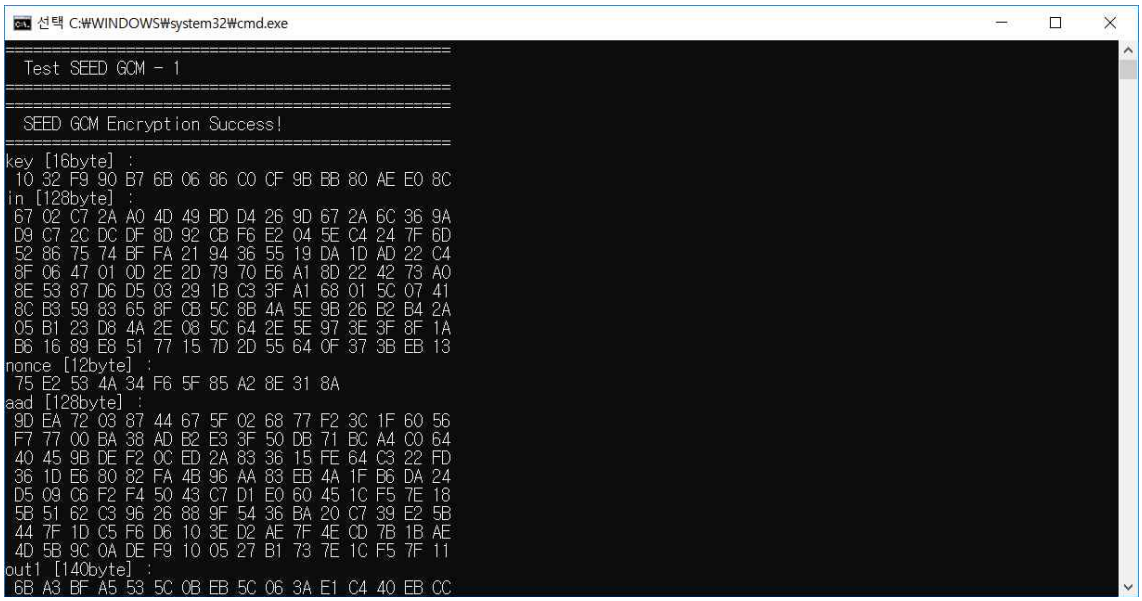
SEED-GCM 알고리즘 MAC 검증 및 복호화 함수

매개변수 :

pt	복호문 출력 버퍼
ptLen	복호문 길이
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :
- 0 : MAC 검증 및 복호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지


선택 C:₩WINDOWS₩system32₩cmd.exe
Test SEED GCM - 1
SEED GCM Encryption Success!
key [16byte] :
10 32 F9 90 B7 6B 06 86 C0 CF 9B BB 80 AE E0 8C
in [128byte] :
67 02 C7 2A A0 4D 49 BD D4 26 9D 67 2A 6C 36 9A
D9 C7 2C DC DF 8D 92 CB F6 E2 04 5E C4 24 7F 6D
52 86 75 74 BF FA 21 94 36 55 19 DA 1D AD 22 C4
8F 06 47 01 0D 2E 2D 79 70 E6 A1 8D 22 42 73 A0
8E 53 87 D6 D5 03 29 1B C3 3F A1 68 01 5C 07 41
8C B3 59 83 65 8F CB 5C 8B 4A 5E 9B 26 B2 B4 2A
05 B1 23 D8 4A 2E 08 5C 64 2E 5E 97 3E 3F 8F 1A
B6 16 89 E8 51 77 15 7D 2D 55 64 0F 37 3B EB 13
nonce [12byte] :
75 E2 53 4A 34 F6 5F 85 A2 8E 31 8A
aad [128byte] :
9D EA 72 03 87 44 67 5F 02 68 77 F2 3C 1F 60 56
F7 77 00 BA 38 AD B2 E3 3F 50 DB 71 BC A4 C0 64
40 45 9B DE F2 0C ED 2A 83 36 15 FE 64 C3 22 FD
36 1D E6 80 82 FA 4B 96 AA 83 EB 4A 1F B6 DA 24
D5 09 C6 F2 F4 50 43 C7 D1 E0 60 45 1C F5 7E 18
5B 51 62 C3 96 26 88 9F 54 36 BA 20 C7 39 E2 5B
44 7F 1D C5 F6 D6 10 3E D2 AE 7F 4E CD 7B 1B AE
4D 5B 9C 0A DE F9 10 05 27 B1 73 7E 1C F5 7F 11
out1 [140byte] :
6B A3 BF A5 53 5C 0B EB 5C 06 3A E1 C4 40 EB CC

바) SEED-CMAC

① 함수 설명

int SEED_Generate_CMAC(unsigned char *pMAC, int macLen, unsigned char *pIn, int inLen, unsigned char *mKey)

SEED-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 16BYTE 보다 큰 경우

int SEED_Verify_CMAC(unsigned char *pMAC, int macLen, unsigned char *pIn, int inLen, unsigned char *mKey)

SEED-CMAC 알고리즘 MAC 검증 함수

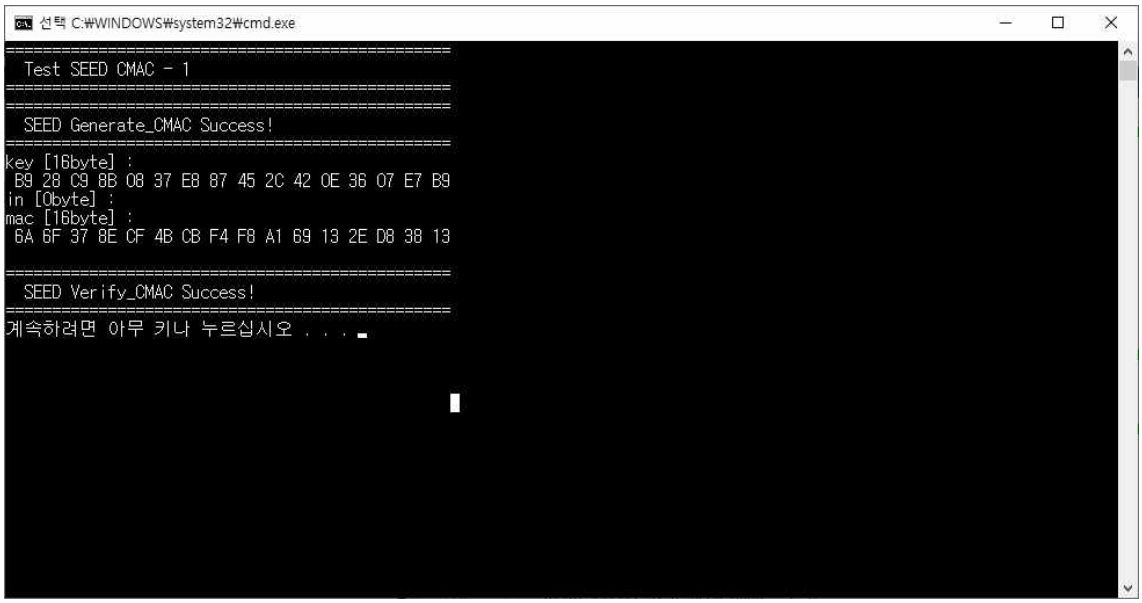
매개변수 :

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 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지

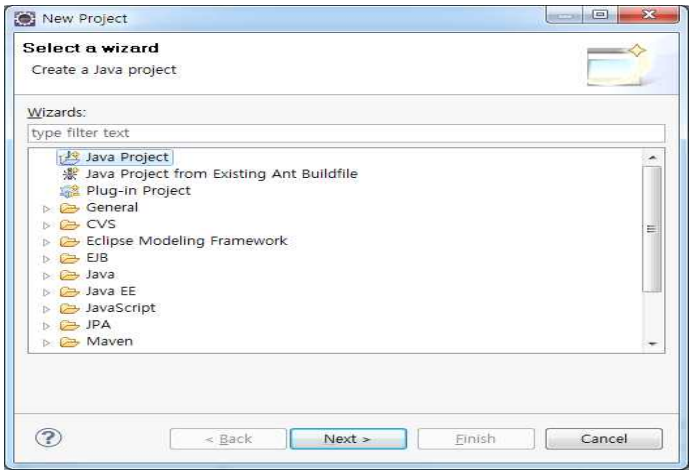

선택 C:₩WINDOWS₩system32₩cmd.exe
Test SEED CMAC - 1
SEED Generate_CMAC Success!
key [16byte] :
B9 28 C9 8B 08 37 E8 87 45 2C 42 0E 36 07 E7 B9
in [0byte] :
mac [16byte] :
6A 6F 37 8E CF 4B CB F4 F8 A1 69 13 2E D8 38 13
SEED Verify_CMAC Success!
계속하려면 아무 키나 누르십시오 . . .

나. Java

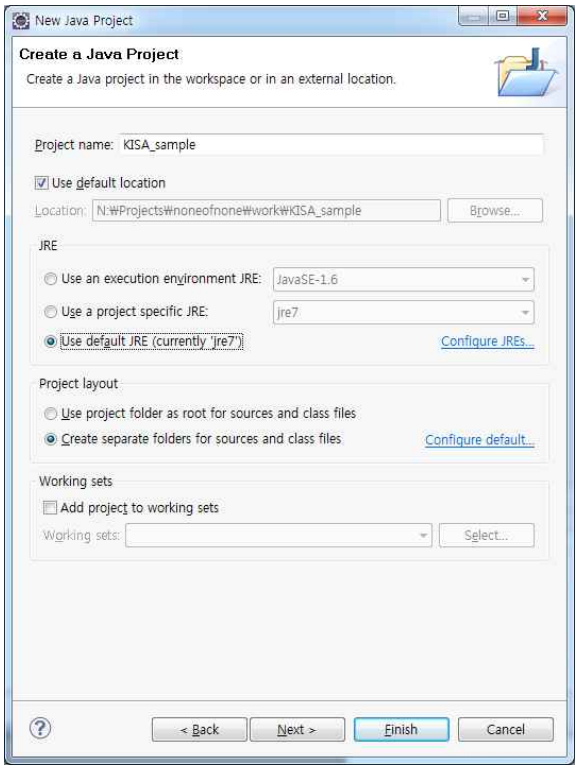
다음은 Java 형태의 프로그램을 하나 생성하여 파일 추가하는 방법과 빌드하는 방법 등을 설명한다.

1) 프로젝트 생성 및 빌드

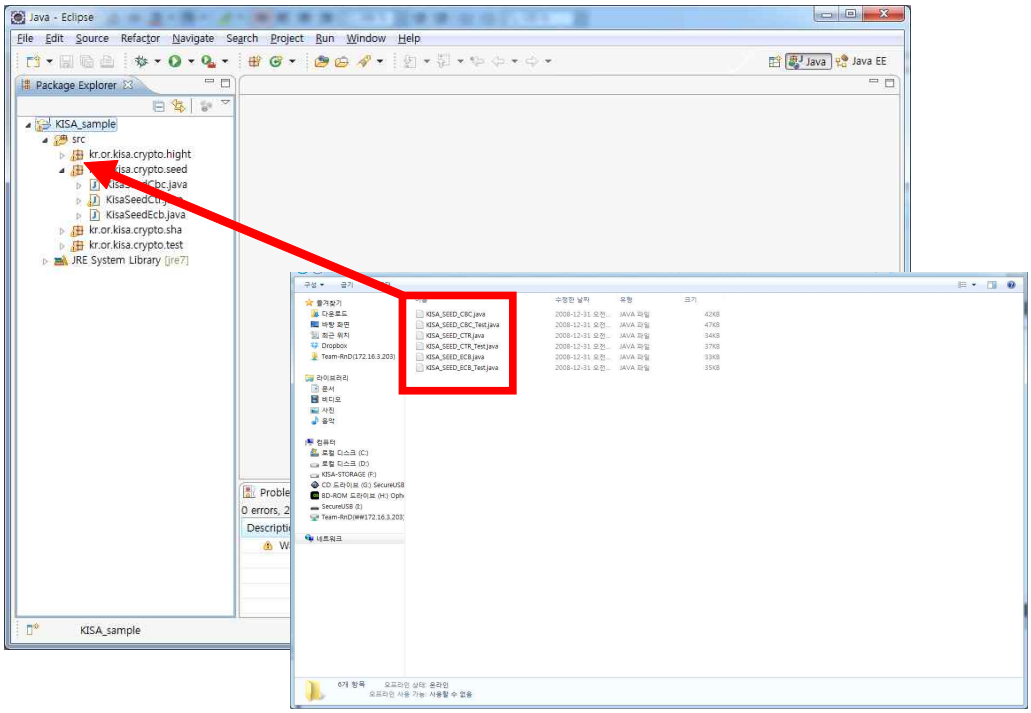
Eclipse를 실행하여 아래의 이미지처럼 Java 프로젝트를 선택한다.



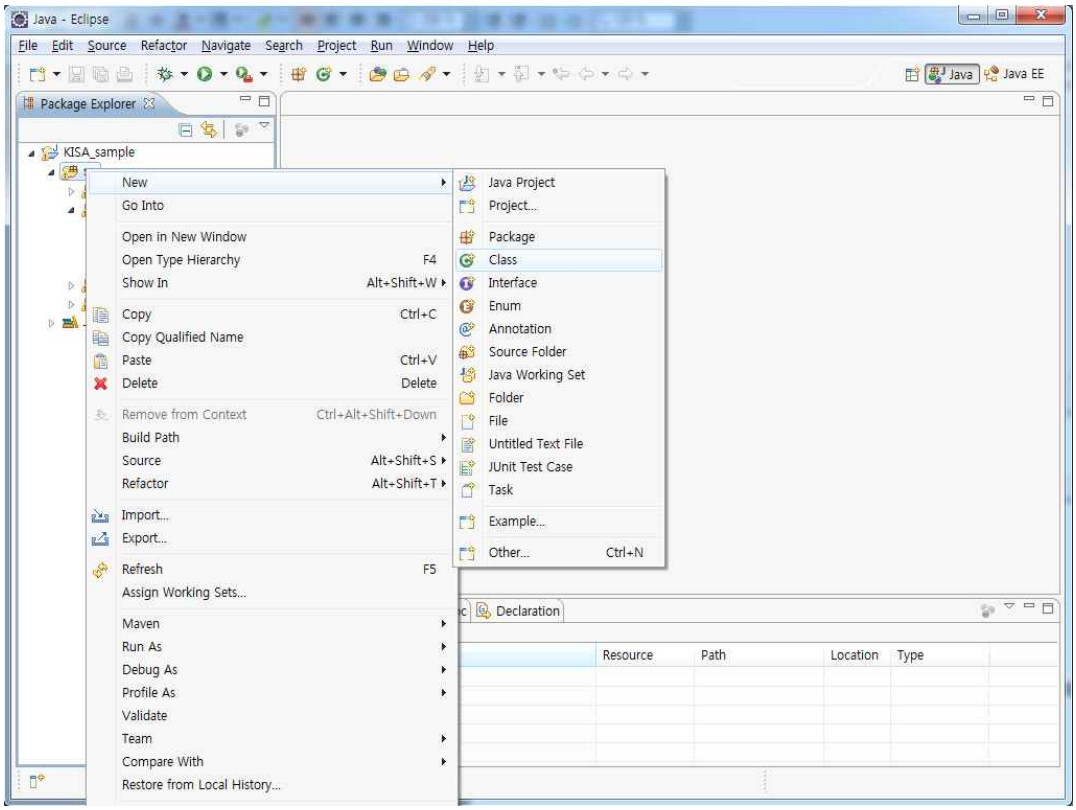
Next를 클릭하여 Project name에는 프로젝트 명을 기입한다. "Use default JRE" 를 선택 후 Finish 버튼을 눌러 프로젝트 생성을 완료한다.



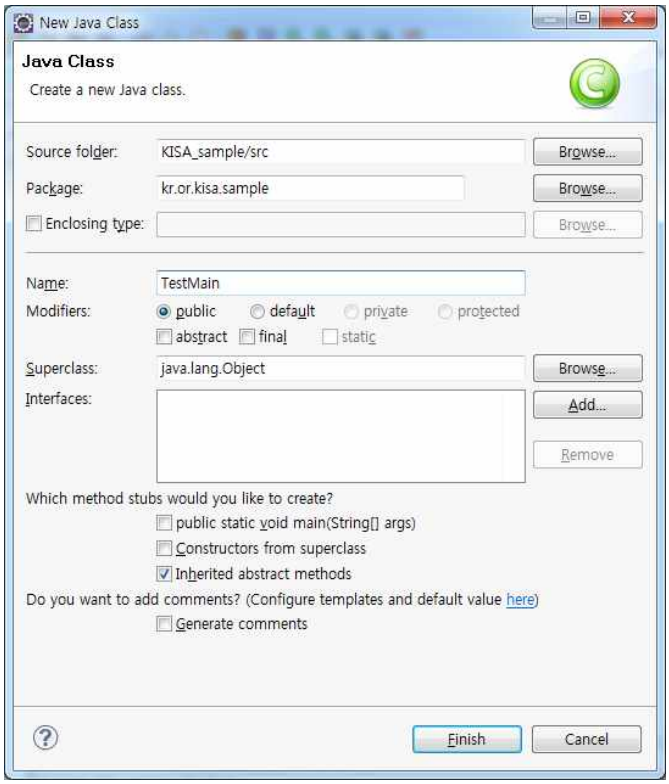
소스가 있는 폴더로 이동하여 프로젝트에 추가할 파일을 드레그&드롭으로 프로젝트로 이동시킨다. 이때 "Copy files and folders"를 선택하여 복사하여 준다.



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

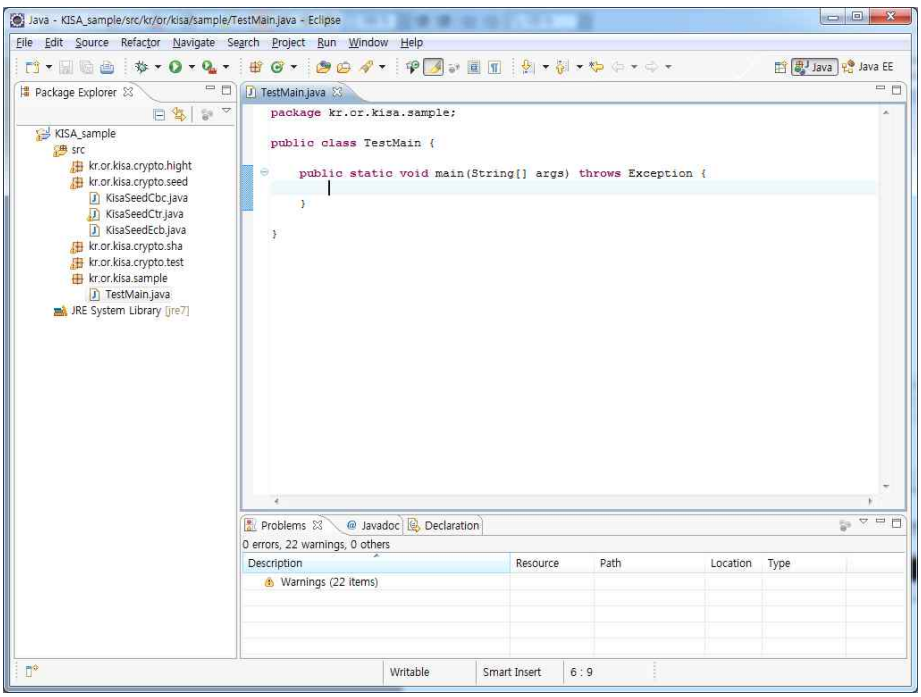


다음으로는 Name을 입력 후 Finish 버튼을 클릭하여 테스트 클래스를 생성한다.



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

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



2) 소스코드 설명

가) SEED-ECB

① 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.	

② 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
```

나) SEED-CBC

① 함수 설명

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

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

② 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
```

다) SEED-CTR

① 함수 설명

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

② 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
```

라) SEED-CCM

① 함수 설명

public int SEED_CCM_Encryption(byte[] ct, byte[] pt, int ptLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)

SEED-CCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0이 아닌 값 : 암호문과 MAC 길이(MAC 생성 및 암호화 성공)
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

public int SEED_CCM_Decryption(byte[] pt, byte[] ct, int ctLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)

SEED-CCM 알고리즘 MAC 검증 및 복호화 함수

매개변수 :

pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0이 아닌 값 : 평문 길이(MAC 검증 및 복호화 성공)
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지

```
================================================
  Test SEED CCM - 1
================================================
================================================
  SEED CCM Encryption Success!
================================================
key [16byte] :
 FA B5 E5 DE 43 50 E5 A4 E0 F1 DF 63 E4 6A 2A A0
in [37byte] :
 E5 46 F3 2B B5 B3 57 40 F3 C4 08 C6 E1 BF 02 53
 09 1C B2 32 DC 94 B9 13 99 7A ED 01 70 4E A0 95
 E8 90 26 69 7E
nonce [12byte] :
 0C 91 14 08 A5 95 DF 62 A9 92 09 C2
aad [32byte] :
 2C 62 D1 FF F6 B7 F6 68 72 66 C2 B3 C7 06 47 36
 44 BA E9 5A 01 4B 1C 4C C3 7A 6F F5 21 94 CA 2D
out1 [53byte] :
 8F 12 A9 B3 5D 42 51 18 01 E8 91 8A C5 E4 F6 56
 8E 82 D8 49 E0 8C E8 49 1C E0 93 E4 75 83 E2 F0
 B3 53 A3 E6 F0 0B 86 87 1A 47 A2 54 C1 FF F7 40
 AB 20 C5 61 44
================================================
  SEED CCM Decryption Success!
================================================
in [53byte] :
 8F 12 A9 B3 5D 42 51 18 01 E8 91 8A C5 E4 F6 56
 8E 82 D8 49 E0 8C E8 49 1C E0 93 E4 75 83 E2 F0
 B3 53 A3 E6 F0 0B 86 87 1A 47 A2 54 C1 FF F7 40
 AB 20 C5 61 44
```

마) SEED-GCM

① 함수 설명

public int SEED_GCM_Encryption(byte[] ct, byte[] pt, int ptLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)

SEED-GCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0이 아닌 값 : 암호문과 MAC 길이(MAC 생성 및 암호화 성공)
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

public int SEED_GCM_Decryption(byte[] pt, byte[] ct, int ctLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)

SEED-GCM 알고리즘 MAC 검증 및 복호화 함수

매개변수 :

pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0이 아닌 값 : 평문 길이(MAC 검증 및 복호화 성공)
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지

```
================================================
  Test SEED GCM - 1
================================================
================================================
  SEED GCM Encryption Success!
================================================
key [16byte] :
 10 32 F9 90 B7 6B 06 86 C0 CF 9B BB 80 AE E0 8C
in [128byte] :
 67 02 C7 2A A0 4D 49 BD D4 26 9D 67 2A 6C 36 9A
 D9 C7 2C DC DF 8D 92 CB F6 E2 04 5E C4 24 7F 6D
 52 86 75 74 BF FA 21 94 36 55 19 DA 1D AD 22 C4
 8F 06 47 01 0D 2E 2D 79 70 E6 A1 8D 22 42 73 A0
 8E 53 87 D6 D5 03 29 1B C3 3F A1 68 01 5C 07 41
 8C B3 59 83 65 8F CB 5C 8B 4A 5E 9B 26 B2 B4 2A
 05 B1 23 D8 4A 2E 08 5C 64 2E 5E 97 3E 3F 8F 1A
 B6 16 89 E8 51 77 15 7D 2D 55 64 0F 37 3B EB 13
nonce [12byte] :
 75 E2 53 4A 34 F6 5F 85 A2 8E 31 8A
aad [128byte] :
 9D EA 72 03 87 44 67 5F 02 68 77 F2 3C 1F 60 56
 F7 77 00 BA 38 AD B2 E3 3F 50 DB 71 BC A4 C0 64
 40 45 9B DE F2 0C ED 2A 83 36 15 FE 64 C3 22 FD
 36 1D E6 80 82 FA 4B 96 AA 83 EB 4A 1F B6 DA 24
 D5 09 C6 F2 F4 50 43 C7 D1 E0 60 45 1C F5 7E 18
 5B 51 62 C3 96 26 88 9F 54 36 BA 20 C7 39 E2 5B
 44 7F 1D C5 F6 D6 10 3E D2 AE 7F 4E CD 7B 1B AE
 4D 5B 9C 0A DE F9 10 05 27 B1 73 7E 1C F5 7F 11
out1 [140byte] :
 6B A3 BF A5 53 5C 0B EB 5C 06 3A E1 C4 40 EB CC
 86 FF 91 A0 9A FA D7 34 97 2E DF AB 10 69 1E B2
```

바) SEED-CMAC

① 함수 설명

public int SEED_Generate_CMAC(byte[] pMAC, int macLen, byte[] pIn, int inLen, byte[] mKey)
SEED-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 16BYTE 보다 큰 경우

public int SEED_Verify_CMAC(byte[] pMAC, int macLen, byte[] pIn, int inLen, byte[] mKey)
SEED-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 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지

```
================================================
  Test SEED CMAC - 1
================================================
================================================
  SEED Generate_CMAC Success!
================================================
key [16byte] :
 B9 28 C9 8B 08 37 E8 87 45 2C 42 0E 36 07 E7 B9
in [0byte] :
mac [16byte] :
 6A 6F 37 8E CF 4B CB F4 F8 A1 69 13 2E D8 38 13
================================================
  SEED Verify_CMAC Success!
================================================
```

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, CTR, CCM, GCM, CMAC이 함께 제공된다.

가. 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.

1) 소스코드 추가

암호화/복호화 소소코드가 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
```

2) 소스코드 설명

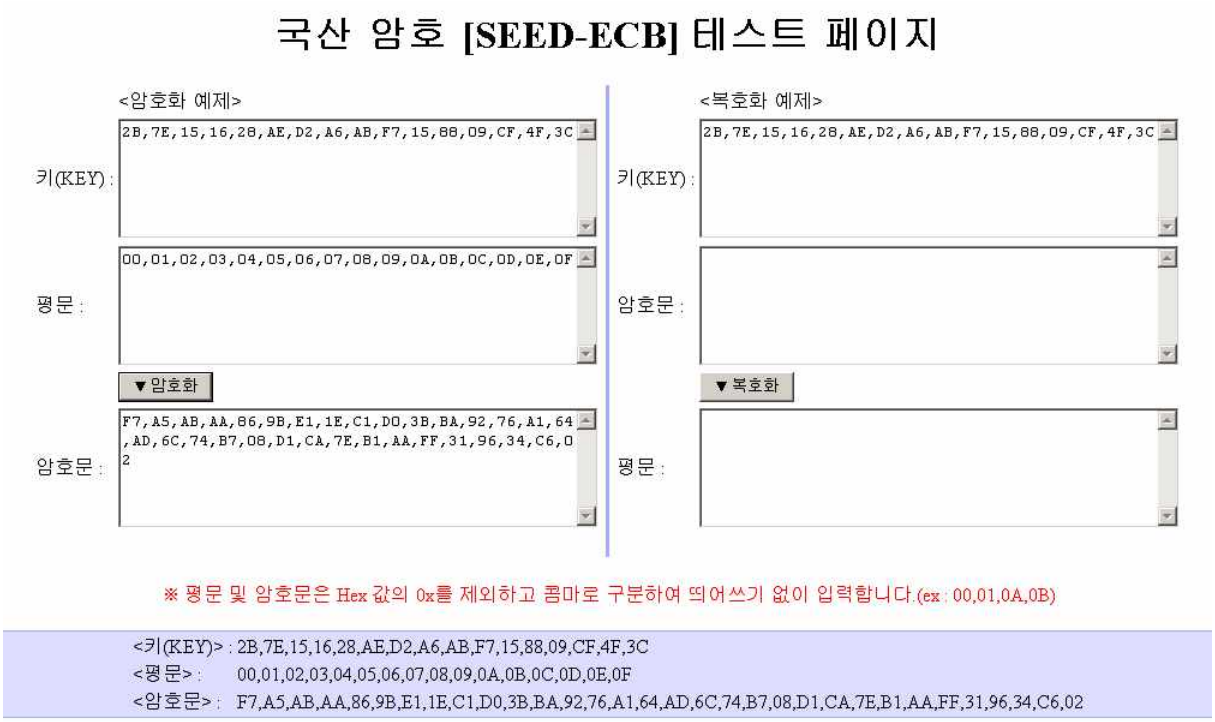
가) SEED-ECB

① 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.

② 테스트 페이지


국산 암호 [SEED-ECB] 테스트 페이지
<암호화 예제>
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
▼암호화
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
<복호화 예제>
키(KEY) :
2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C
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

나) SEED-CBC

① 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

② 테스트 페이지


국산 암호 [SEED-CBC] 테스트 페이지
<암호화 예제>
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
▼암호화
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
<복호화 예제>
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
암호문 :
▼복호화
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

다) SEED-CTR

① 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

② 테스트 페이지

라) SEED-CCM

① 함수 설명

public function SEED_CCM_Encryption(byref ct, pt, ptLen, macLen, nonce, nonceLen, aad, aadLen, mKey)

SEED-CCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 생성 및 암호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

public function SEED_CCM_Decryption(byref pt, ct, ctLen, macLen, nonce, nonceLen, aad, aadLen, mKey)

SEED-CCM 알고리즘 MAC 검증 및 복호화 함수

매개변수 :

pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 검증 및 복호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지

국산 암호 [SEED-CCM] 테스트 페이지

<암호화 예제>

키(KEY) : FC, 58, 7C, 16, 26, 93, E6, CD, 63, EE, D5, 39, B5, 7B, EA, 09

초기값(NONCE) : 5C, 85, 10, 0A, 3E, 69, 01

추가인증데이터(AAD) : 9D, 8C, A7, 0D, 69, A3, 39, 17, 4D, 30, 24, E0, 98, 98, 4C, 88

평문 : 7D, 97, 8C, 51, C1, 27, 06, A7, B7, A3, B8, 5D, 6E, 2C, 51, 3A

인증값 길이 : 16

▼ 암호화

암호문 : 47, 71, D9, F2, 50, 3C, BF, EB, B2, 00, CB, ED, 10, 22, 42, EC, AA, CD, 3A, 5F, 54, 84, 86, 3C, AF, 97, 18, 8D, 7B, 67, D0, 05

결과값 : 0, Success!

<암호화 예제>

키(KEY) :

초기값(NONCE) :

추가인증데이터(AAD) :

암호문 :

인증값 길이 :

▼ 복호화

복호문 :

결과값 :

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

<키(KEY)> : FC,58,7C,16,26,93,E6,CD,63,EE,D5,39,B5,7B,EA,09
<초기값(NONCE)> : 5C,85,10,0A,3E,69,01
<추가인증데이터(AAD)> : 9D,8C,A7,0D,69,A3,39,17,4D,30,24,E0,98,98,4C,88
<평문> : 7D,97,8C,51,C1,27,06,A7,B7,A3,B8,5D,6E,2C,51,3A
<인증값 길이> : 16
<암호문> : 47,71,D9,F2,50,3C,BF,EB,B2,00,CB,ED,10,22,42,EC,AA,CD,3A,5F,54,84,86,3C,AF,97,18,8D,7B,67,D0,05

마) SEED-GCM

① 함수 설명

public function SEED_GCM_Encryption(byref ct, pt, ptLen, macLen, nonce, nonceLen, aad, aadLen, mKey)

SEED-GCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 생성 및 암호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

public function SEED_GCM_Decryption(byref pt, ct, ctLen, macLen, nonce, nonceLen, aad, aadLen, mKey)

SEED-GCM 알고리즘 MAC 검증 및 복호화 함수

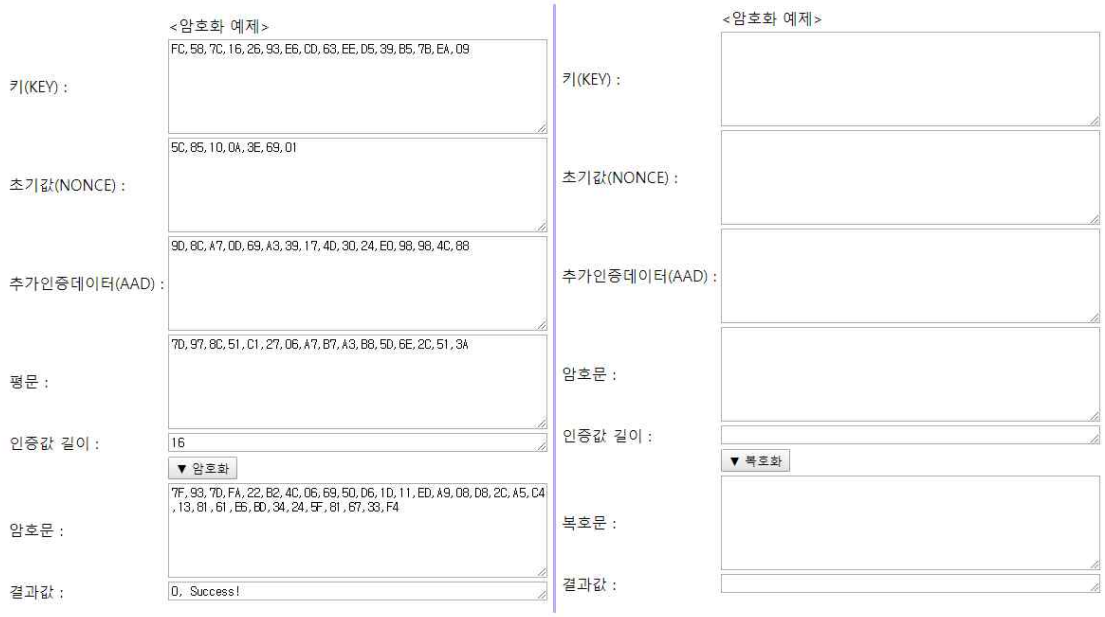
매개변수 :

pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 검증 및 복호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지



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

<키(KEY)> :	FC,58,7C,16,26,93,E6,CD,63,EE,D5,39,B5,7B,EA,09
<초기값(NONCE)> :	5C,85,10,0A,3E,69,01
<추가인증데이터(AAD)> :	9D,8C,A7,0D,69,A3,39,17,4D,30,24,E0,98,98,4C,88
<평문> :	7D,97,8C,51,C1,27,06,A7,B7,A3,B8,5D,6E,2C,51,3A
<인증값 길이> :	16
<암호문> :	7F,93,7D,FA,22,B2,4C,06,69,50,D6,1D,11,ED,A9,08,D8,2C,A5,C4,13,81,61,E6,BD,34,24,5F,81,67,33,F4

바) SEED-CMAC

① 함수 설명

public function SEED_Generate_CMAC(byref pMAC, macLen, pIn, inLen, mKey)

SEED-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 16BYTE 보다 큰 경우

public function SEED_Verify_CMAC(byref pMAC, macLen, pIn, inLen, mKey)

SEED-CMAC 알고리즘 MAC 검증 함수

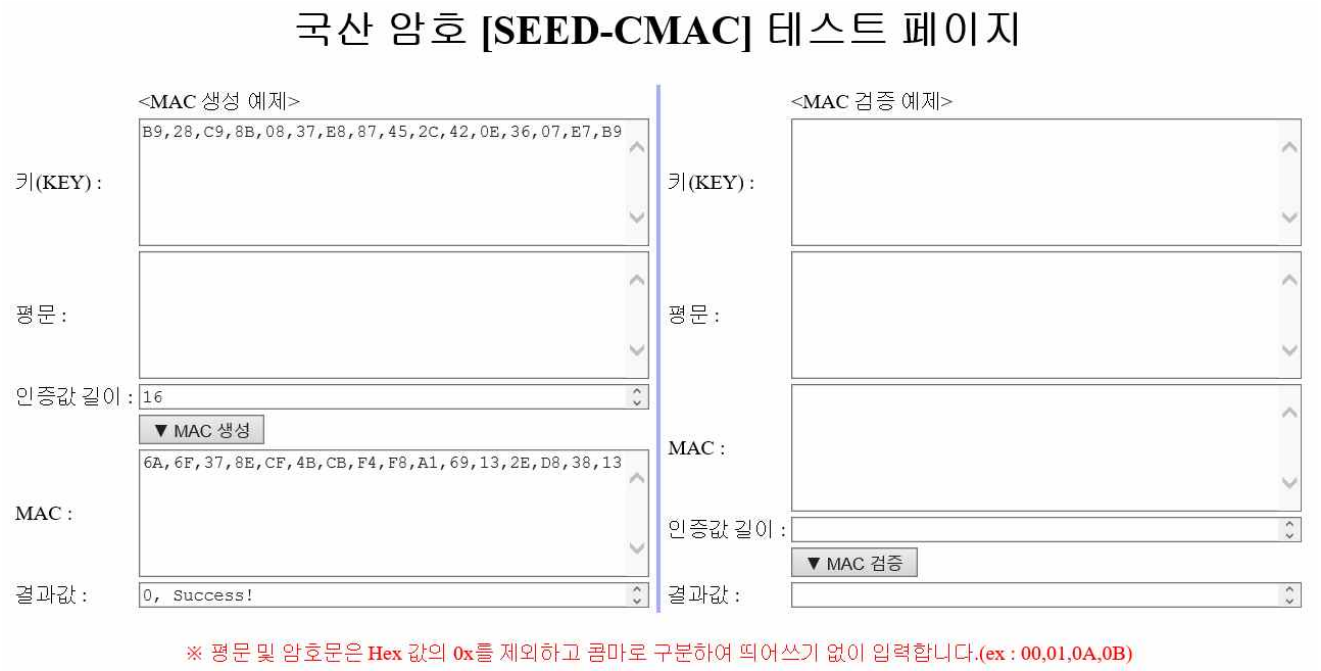
매개변수 :

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 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지


국산 암호 [SEED-CMAC] 테스트 페이지
<MAC 생성 예제>
키(KEY) :
B9,28,C9,8B,08,37,E8,87,45,2C,42,0E,36,07,E7,B9
평문 :
인증값 길이 :
16
▼ MAC 생성
MAC :
6A,6F,37,8E,CF,4B,CB,F4,F8,A1,69,13,2E,D8,38,13
결과값 :
0, Success!
<MAC 검증 예제>
키(KEY) :
평문 :
MAC :
인증값 길이 :
▼ MAC 검증
결과값 :
※ 평문 및 암호문은 Hex 값의 0x를 제외하고 콤마로 구분하여 띄어쓰기 없이 입력합니다.(ex : 00,01,0A,0B)
<키(KEY)> : B9,28,C9,8B,08,37,E8,87,45,2C,42,0E,36,07,E7,B9
<평문> :
<인증값 길이> : 16
<MAC> : 6A,6F,37,8E,CF,4B,CB,F4,F8,A1,69,13,2E,D8,38,13

나. 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.

1) 소스코드 추가

암호화/복호화 소소코드가 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;
}
```

2) 소스코드 설명

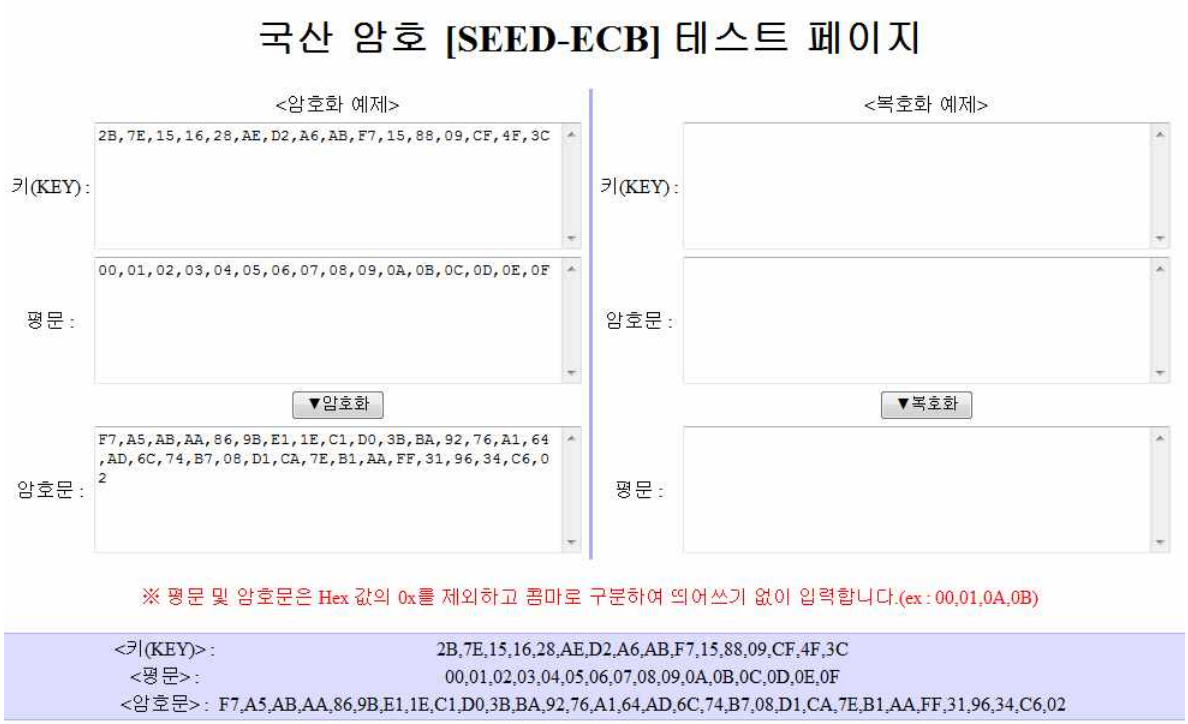
가) SEED-ECB

① 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.

② 테스트 페이지


국산 암호 [SEED-ECB] 테스트 페이지
<암호화 예제>
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
▼암호화
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,C6,0
2
<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

나) SEED-CBC

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

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 블럭(16 bytes)이상으로 메모리가 할당되어 있어야 한다.

② 테스트 페이지

다) SEED-CTR

① 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 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 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

② 테스트 페이지

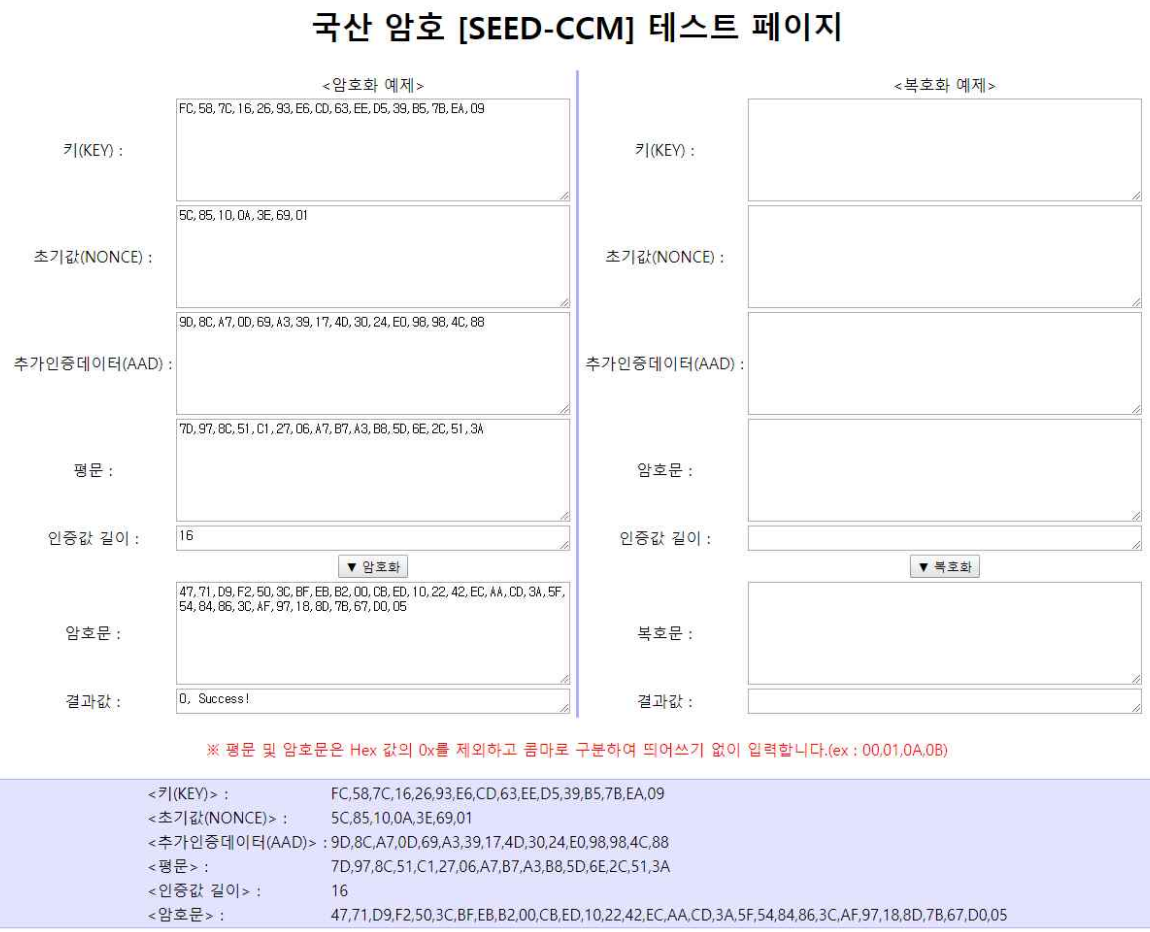
라) SEED-CCM

① 함수 설명

public static int SEED_CCM_Encryption(byte[] ct, byte[] pt, int ptLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)	
SEED-CCM 알고리즘 MAC 생성 및 암호화 함수	
매개변수 :	
ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)
반환값 :	
- 0 : MAC 생성 및 암호화 성공	
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우	

public static int SEED_CCM_Decryption(byte[] pt, byte[] ct, int ctLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)	
SEED-CCM 알고리즘 MAC 검증 및 복호화 함수	
매개변수 :	
pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)
반환값 :	
- 0 : MAC 검증 및 복호화 성공	
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우	

② 테스트 페이지



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

<키(KEY)> : FC,58,7C,16,26,93,E6,CD,63,EE,D5,39,B5,7B,EA,09
<초기값(NONCE)> : 5C,85,10,0A,3E,69,01
<추가인증데이터(AAD)> : 9D,8C,A7,0D,69,A3,39,17,4D,30,24,E0,98,98,4C,88
<평문> : 7D,97,8C,51,C1,27,06,A7,B7,A3,B8,5D,6E,2C,51,3A
<인증값 길이> : 16
<암호문> : 47,71,D9,F2,50,3C,BF,EB,B2,00,CB,ED,10,22,42,EC,AA,CD,3A,5F,54,84,86,3C,AF,97,18,8D,7B,67,D0,05

마) SEED-GCM

① 함수 설명

public static int SEED_GCM_Encryption(byte[] ct, byte[] pt, int ptLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)	
SEED-GCM 알고리즘 MAC 생성 및 암호화 함수	
매개변수 :	
ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)
반환값 :	
- 0 : MAC 생성 및 암호화 성공	
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우	

public static int SEED_GCM_Decryption(byte[] pt, byte[] ct, int ctLen, int macLen, byte[] nonce, int nonceLen, byte[] aad, int aadLen, byte[] mKey)	
SEED-GCM 알고리즘 MAC 검증 및 복호화 함수	
매개변수 :	
pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)
반환값 :	
- 0 : MAC 검증 및 복호화 성공	
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우	

② 테스트 페이지

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

<키(KEY)> :	FC,58,7C,16,26,93,E6,CD,63,EE,D5,39,B5,7B,EA,09
<초기값(NONCE)> :	5C,85,10,0A,3E,69,01
<추가인증데이터(AAD)> :	9D,8C,A7,0D,69,A3,39,17,4D,30,24,E0,98,98,4C,88
<평문> :	7D,97,8C,51,C1,27,06,A7,B7,A3,B8,5D,6E,2C,51,3A
<인증값 길이> :	16
<암호문> :	7F,93,7D,FA,22,B2,4C,06,69,50,D6,1D,11,ED,A9,08,D8,2C,A5,C4,13,81,61,E6,BD,34,24,5F,81,67,33,F4

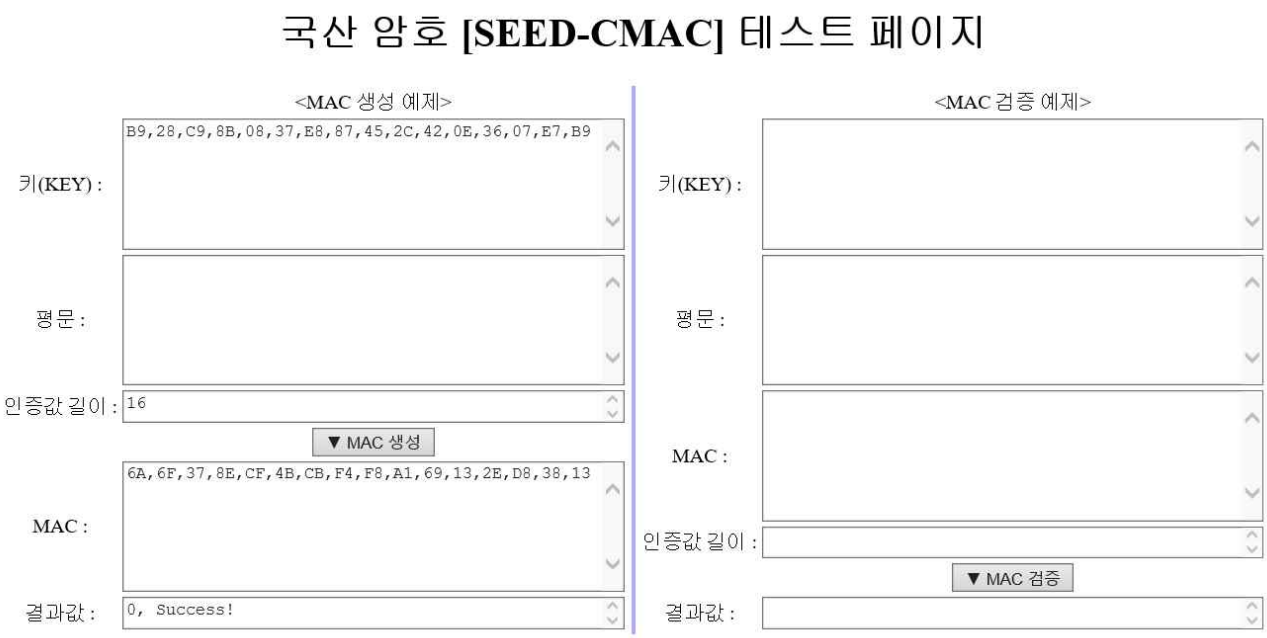
바) SEED-CMAC

① 함수 설명

public static int SEED_Generate_CMAC(byte[] pMAC, int macLen, byte[] pIn, int inLen, byte[] mKey)
SEED-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 16BYTE 보다 큰 경우

public static int SEED_Verify_CMAC(byte[] pMAC, int macLen, byte[] pIn, int inLen, byte[] mKey)
SEED-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 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지



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

<키(KEY)> : B9,28,C9,8B,08,37,E8,87,45,2C,42,0E,36,07,E7,B9
<평문> :
<인증값 길이> : 16
<암호문> : 6A,6F,37,8E,CF,4B,CB,F4,F8,A1,69,13,2E,D8,38,13

다. 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.

1) 소스코드 추가

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

```
<?
require_once ('KISA_SEED_ECB.php');

$g_bszUser_key = $_POST['KEY'];

if($g_bszUser_key == null)
{
    $g_bszUser_key = "2b,7e,15,16,28,ae,d2,a6,ab,f7,15,88,09,cf,4f,3c";
}

function encrypt($bszUser_key, $str) {


    $planBytes = split(",",$str);
    $keyBytes = split(",",$bszUser_key);

    for($i = 0; $i < 16; $i++)
    {
        $keyBytes[$i] = hexdec($keyBytes[$i]);
    }
    for ($i = 0; $i < count($planBytes); $i++) {
        $planBytes[$i] = hexdec($planBytes[$i]);
    }

    if (count($planBytes) == 0) {
        return $str;
    }
```

2) 소스코드 설명

가) SEED-ECB

① 함수 설명

static function SEED_ECB_Encrypt(&$pbszUserKey, &$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.

static function SEED_ECB_Decrypt(&$pbszUserKey, &$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.

② 테스트 페이지

국산 암호 [SEED-ECB] 테스트 페이지

<암호화 예제>

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

▼ 암호화

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

<복호화 예제>

키(KEY) : 2b,7e,15,16,28,ae,d2,a6,ab,f7,15,88,09,cf,4f,3c

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

나) SEED-CBC

① 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

② 테스트 페이지


국산 암호 [SEED-CBC] 테스트 페이지
<암호화 예제>
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
▼ 암호화
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
<복호화 예제>
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
암호문 :
▼ 복호화
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

다) SEED-CTR

① 함수 설명

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	

② 테스트 페이지

라) SEED-CCM

① 함수 설명

static function SEED_CCM_Encryption(&$ct, $pt, $ptLen, $macLen, $nonce, $nonceLen, $aad, $aadLen, $mKey)

SEED-CCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 생성 및 암호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

static function SEED_CCM_Decryption(&$pt, $ct, $ctLen, $macLen, $nonce, $nonceLen, $aad, $aadLen, $mKey)

SEED-CCM 알고리즘 MAC 검증 및 복호화 함수

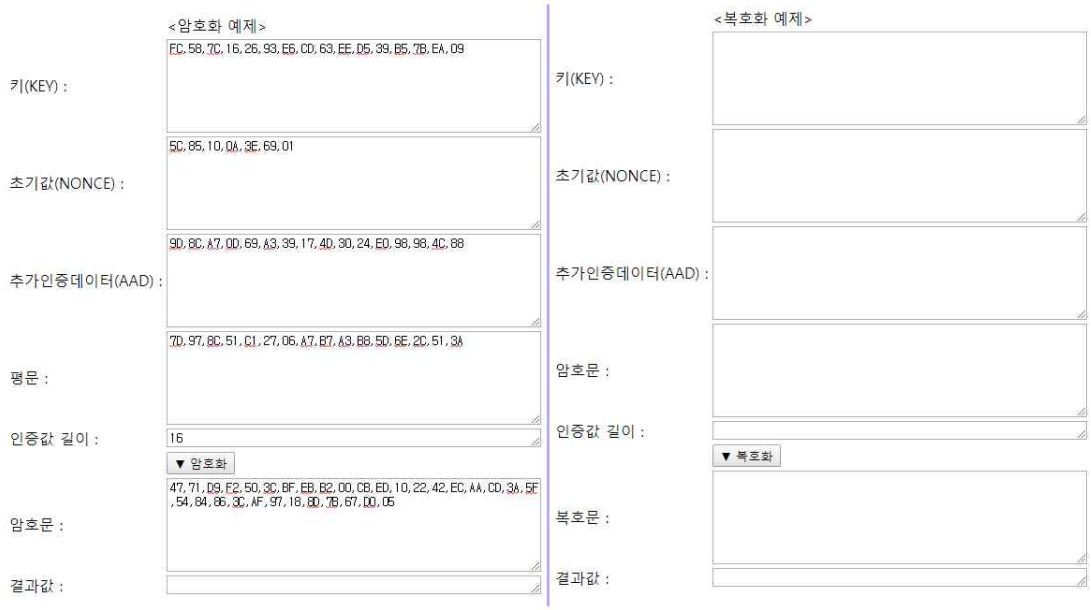
매개변수 :

pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 검증 및 복호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지



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

<키(KEY)> :	FC,58,7C,16,26,93,E6,CD,63,EE,D5,39,B5,7B,EA,09
<초기값(NONCE)> :	5C,85,10,0A,3E,69,01
<추가인증데이터(AAD)> :	9D,8C,A7,0D,69,A3,39,17,4D,30,24,E0,98,98,4C,88
<평문> :	7D,97,8C,51,C1,27,06,A7,B7,A3,B8,5D,6E,2C,51,3A
<인증값 길이> :	16
<암호문> :	47,71,D9,F2,50,3C,BF,EB,B2,00,CB,ED,10,22,42,EC,AA,CD,3A,5F,54,84,86,3C,AF,97,18,8D,7B,67,D0,05

마) SEED-GCM

① 함수 설명

static function SEED_GCM_Encryption(&$ct, $pt, $ptLen, $macLen, $nonce, $nonceLen, $aad, $aadLen, $mKey)

SEED-GCM 알고리즘 MAC 생성 및 암호화 함수

매개변수 :

ct	암호문과 MAC 출력 버퍼
pt	사용자 입력 평문
ptLen	평문 길이(BYTE 단위의 평문 길이)
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 생성 및 암호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 큰 경우

static function SEED_GCM_Decryption(&$pt, $ct, $ctLen, $macLen, $nonce, $nonceLen, $aad, $aadLen, $mKey)

SEED-GCM 알고리즘 MAC 검증 및 복호화 함수

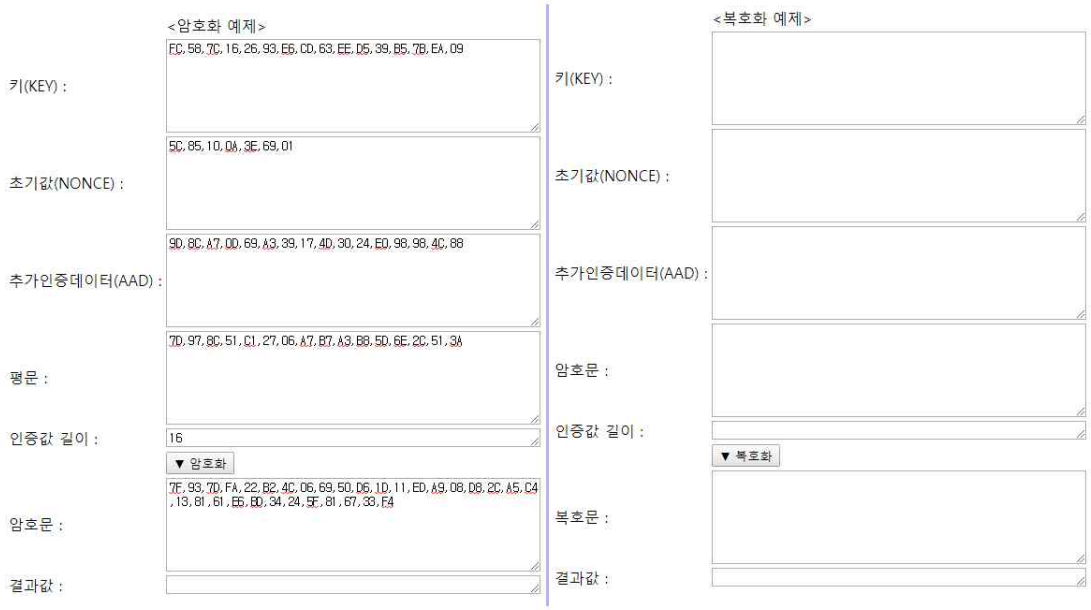
매개변수 :

pt	복호문 출력 버퍼
ct	암호문과 MAC 입력 데이터
ctLen	암호문과 MAC 입력 데이터 길이
macLen	MAC 길이(BYTE 단위의 MAC 길이)
nonce	Nonce
nonceLen	Nonce 길이(BYTE 단위의 Nonce 길이)
aad	부가 인증 데이터
aadLen	부가 인증 데이터 길이(BYTE 단위의 부가 인증 데이터 길이)
mKey	사용자가 지정하는 입력 키(16 BYTE)

반환값 :

- 0 : MAC 검증 및 복호화 성공
- 1 : MAC 길이가 블록 크기인 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지



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

<키(KEY)> :	FC,58,7C,16,26,93,E6,CD,63,EE,D5,39,B5,7B,EA,09
<초기값(NONCE)> :	5C,85,10,0A,3E,69,01
<추가인증데이터(AAD)> :	9D,8C,A7,0D,69,A3,39,17,4D,30,24,E0,98,98,4C,88
<평문> :	7D,97,8C,51,C1,27,06,A7,B7,A3,B8,5D,6E,2C,51,3A
<인증값 길이> :	16
<암호문> :	7F,93,7D,FA,22,B2,4C,06,69,50,D6,1D,11,ED,A9,08,D8,2C,A5,C4,13,81,61,E6,BD,34,24,5F,81,67,33,F4

바) SEED-CMAC

① 함수 설명

static function SEED_Generate_CMAC(&$pMAC, $macLen, $pIn, $inLen, $mKey)

SEED-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 16BYTE 보다 큰 경우

static function SEED_Verify_CMAC($pMAC, $macLen, $pIn, $inLen, $mKey)

SEED-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 16BYTE 보다 크거나 MAC 검증에 실패한 경우

② 테스트 페이지

국산 암호 [SEED-CMAC] 테스트 페이지

<MAC 생성 예제>

키(KEY) : B9,28,C9,8B,08,37,E8,87,45,2C,42,0E,36,07,E7,B9

평문 :

인증값 길이 : 16

▼ MAC 생성

MAC : 6A,6F,37,8E,CF,4B,CB,F4,F8,A1,69,13,2E,D8,38,13

결과값 : 0, Success!

<MAC 검증 예제>

키(KEY) :

평문 :

MAC :

인증값 길이 :

▼ MAC 검증

결과값 :

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

<키(KEY)> : B9,28,C9,8B,08,37,E8,87,45,2C,42,0E,36,07,E7,B9
<평문> :
<인증값 길이> : 16
<MAC> : 6A,6F,37,8E,CF,4B,CB,F4,F8,A1,69,13,2E,D8,38,13

6. 참조구현값

본 SEED 소스코드 매뉴얼에서는 ECB, CBC, CTR, CCM, GCM, 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(512비트)에 대한 참조구현값은 암·복호화 연산에서 단계별 입력 블록(I_i), 출력 블록(O_i)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 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

CCM, GCM 운영모드에 대한 참조구현값을 생성하기 위한 평문 데이터와 키, 초기값, 초기 카운트는 아래 표와 같다. 데이터 3(296비트)에 대한 참조구현값은 CCM 운영모드에 대한 참조값으로 평문 데이터를 이용하여 MAC값과 암호문 생성하기 위한 중간값과, 이를 통해 MAC값을 검증하고 복호화하기 위한 중간값들을 제공한다. 데이터 3(1,024비트)에 대한 참조구현값은 GCM 운영모드에 대한 참조값으로 평문 데이터를 이용하여 MAC값과 암호문 생성하기 위한 중간값과, 이를 통해 MAC값을 검증하고 복호

화하기 위한 중간값들을 제공한다.

< 데이터 3 - 296비트 >

키(Key)	47 86 F7 50 27 CA 81 B9 B7 1C 67 C4 66 81 2B BB
난스(Nonce)	65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C
부가 인증 데이터	EB 6A 52 DD 95 81 9C D2 B4 C0 4D C1 01 2C 30 30 B1 4A DC 7F C2 49 BD DD B9 EE 0F 92 58 90 6F 55
평문(P)	51 22 6C BF 80 A3 D3 91 B2 E7 F1 FB 85 1B B2 76 66 EB BF 73 31 B7 89 6F 88 BF 65 AD 5A A4 A1 21 AA 38 C5 D0 1D
MAC 길이	16 byte

< 데이터 4 - 1,024비트 >

키(Key)	D5 9F BC 68 0C 17 42 46 5F 9A 87 53 EA 25 9F 1C
난스(Nonce)	93 58 BE DC 3E D6 06 FC 40 EF 9C D6 16 7D 01 BE
부가 인증 데이터	91 AA E5 0D BC 2A 9E 7B E5 93 F6 F5 30 C7 5D FB AD E5 80 58 8A B6 1A A6 87 48 0D 14 A9 D8 B1 67 33 F5 6C 3F D2 E3 83 9C 18 3E 2F A0 2B DF F4 97 BF 04 2E 34 41 AA ED 1D 55 3E 02 AB 83 75 5A EA 2D 79 8B E8 C5 45 AD 2C 3D 52 6B 84 BE 73 59 04 9C FD 89 4B 8A 2C 59 08 0A C2 8E BD 28 F1 A7 C3 68 7B 6B 8F FD 18 C5 33 3B 18 D1 27 4F 49 37 4A 2E 1B B8 24 CB 02 06 6E 8D 1C D9 D4 00 14 3F F9
평문(P)	58 0B A0 43 E8 3D A3 B2 6F 6C 0B DC 0A 90 00 19 40 63 65 8D 38 C3 B0 AA 28 58 F9 F6 11 78 50 0F AA 47 54 EC 99 23 02 65 E9 3F CA F6 B9 8D E8 C0 31 E2 F1 D0 BB 55 AC A4 6F E8 24 EB CE 6A FB 8D B4 9B 01 EB 88 92 05 68 B6 5C EC 28 5F E6 00 18 4F 1C 8A 2E 2F 53 A2 F2 FB E4 46 3D B7 1A AA 40 5E 4F CF C9 1C 51 57 C2 BC 0B 98 FA 64 61 EF 29 80 5B 56 2E FC 86 39 9A B6 97 86 72 33 52 04 31
MAC 길이	4 byte

CMAC 운영모드에 대한 참조구현값을 생성하기 위한 평문 데이터, 키는 아래 표와 같다. 데이터 5에 대한 참조구현값은 평문 데이터를 이용하여 MAC값을 생성하기 위한 중간값들을 제공한다.

< 데이터 5 >

키(Key)	B7 28 C9 8B 08 37 E8 87 45 2C 42 0E 36 07 E7 B9
평문(P)	(NULL)
MAC 길이	16 byte

가. ECB 모드 참조구현값

1) 데이터 1 - 암호화

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

2) 데이터 1 - 복호화

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

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(C0)	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

4) 서명 검증 과정

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

나. CBC 모드 참조구현값

1) 데이터 1 - 암호화

블록1	평문(P1)	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
	입력(I1)	F1 E0 6B BF 07 D6 DF E3 DE E4 B2 39 53 BA 29 0E
	출력(01)	A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27
	암호문(C1)	A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27
블록2	평문(P2)	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
	입력(I2)	2F D2 0A 52 4A 2B D7 02 DC 47 E3 A6 CD 8B 44 B1
	출력(02)	E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34
	암호문(C2)	E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34
블록3	평문(P3)	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
	입력(I3)	D1 FA 26 2B B5 AE 7D 6B 6F 6E 89 1C 57 C0 10 A7
	출력(03)	CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79
	암호문(C3)	CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79
블록4	평문(P4)	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3
	입력(I4)	1A 7F 56 E6 89 9A 6D B9 59 25 56 D5 64 DC 4A CA
	출력(04)	C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06
	암호문(C4)	C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06

2) 데이터 1 - 복호화

블록1	암호문(C1)	A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27
	입력(I1)	A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27
	출력(01)	F1 E0 6B BF 07 D6 DF E3 DE E4 B2 39 53 BA 29 0E
	평문(P1)	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
블록2	암호문(C2)	E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34
	입력(I2)	E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34
	출력(02)	2F D2 0A 52 4A 2B D7 02 DC 47 E3 A6 CD 8B 44 B1
	평문(P2)	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
블록3	암호문(C3)	CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79
	입력(I3)	CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79
	출력(03)	D1 FA 26 2B B5 AE 7D 6B 6F 6E 89 1C 57 C0 10 A7
	평문(P3)	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
블록4	암호문(C4)	C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06
	입력(I4)	C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06
	출력(04)	1A 7F 56 E6 89 9A 6D B9 59 25 56 D5 64 DC 4A CA
	평문(P4)	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3

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

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

다. CTR 모드 참조구협값

1) 데이터 1 - 암호화

블록1	평문(P1)	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
	입력(I1)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE
	출력(01)	83 73 11 DC 65 D8 CD 5C 58 A5 E1 5F 57 8B 4D E2
	암호문(C1)	54 1E 1C C4 57 A6 08 3E E9 FB 8A 9C 32 27 41 ED
블록2	평문(P2)	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
	입력(I2)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF
	출력(02)	19 7B CC 56 B6 DF F4 9E 13 2C AC FD 28 75 55 D3
	암호문(C2)	94 3A 2C ED 25 5A 9C 30 F8 D1 3E 10 32 8A F5 45
블록3	평문(P3)	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
	입력(I3)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00
	출력(03)	15 1F FD 7E 39 93 2C 79 5B 0F 8D 05 FE 27 30 C8
	암호문(C3)	2C 52 DD 82 6B E4 F1 85 16 E7 3D F9 1F CC 1B 5B
블록4	평문(P4)	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3
	입력(I4)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01
	출력(04)	0F A3 40 18 62 4C 97 1B 96 77 E2 51 A7 31 4C B6
	암호문(C4)	DB 0D 00 F7 25 24 51 08 23 7C 6B 13 50 E5 F5 05

2) 데이터 1 - 복호화

블록1	암호문(C1)	54 1E 1C C4 57 A6 08 3E E9 FB 8A 9C 32 27 41 ED
	입력(I1)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE
	출력(01)	83 73 11 DC 65 D8 CD 5C 58 A5 E1 5F 57 8B 4D E2
	평문(P1)	D7 6D 0D 18 32 7E C5 62 B1 5E 6B C3 65 AC 0C 0F
블록2	암호문(C2)	94 3A 2C ED 25 5A 9C 30 F8 D1 3E 10 32 8A F5 45
	입력(I2)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF
	출력(02)	19 7B CC 56 B6 DF F4 9E 13 2C AC FD 28 75 55 D3
	평문(P2)	8D 41 E0 BB 93 85 68 AE EB FD 92 ED 1A FF A0 96
블록3	암호문(C3)	2C 52 DD 82 6B E4 F1 85 16 E7 3D F9 1F CC 1B 5B
	입력(I3)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00
	출력(03)	15 1F FD 7E 39 93 2C 79 5B 0F 8D 05 FE 27 30 C8
	평문(P3)	39 4D 20 FC 52 77 DD FC 4D E8 B0 FC E1 EB 2B 93
블록4	암호문(C4)	DB 0D 00 F7 25 24 51 08 23 7C 6B 13 50 E5 F5 05
	입력(I4)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01
	출력(04)	0F A3 40 18 62 4C 97 1B 96 77 E2 51 A7 31 4C B6
	평문(P4)	D4 AE 40 EF 47 68 C6 13 B5 0B 89 42 F7 D4 B9 B3

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(C0)	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

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

라. CCM 모드 참조구협값

1) 데이터 3 - 암호화

입력	키(Key)			47 86 F7 50 27 CA 81 B9 B7 1C 67 C4 66 81 2B BB
	난스(Nonce)			65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C
	부가 인증 데이터			EB 6A 52 DD 95 81 9C D2 B4 C0 4D C1 01 2C 30 30 B1 4A DC 7F C2 49 BD DD B9 EE 0F 92 58 90 6F 55
	평문(P)			51 22 6C BF 80 A3 D3 91 B2 E7 F1 FB 85 1B B2 76 66 EB BF 73 31 B7 89 6F 88 BF 65 AD 5A A4 A1 21 AA 38 C5 D0 1D
	MAC 길이			16 byte
처리 과정	B			7A 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 25 00 20 EB 6A 52 DD 95 81 9C D2 B4 C0 4D C1 01 2C 30 30 B1 4A DC 7F C2 49 BD DD B9 EE 0F 92 58 90 6F 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 51 22 6C BF 80 A3 D3 91 B2 E7 F1 FB 85 1B B2 76 66 EB BF 73 31 B7 89 6F 88 BF 65 AD 5A A4 A1 21 AA 38 C5 D0 1D 00 00 00 00 00 00 00 00 00 00 00
	MAC값 계산	2-0	B_0	7A 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 25
			Y	08 92 E8 7E 03 4D 47 52 FF 23 22 10 B2 6F A3 09
		2-1	X	08 B2 03 14 51 90 D2 D3 63 F1 96 D0 FF AE A2 25
			Y	0C 4E 9D E1 21 00 C6 1E ED 25 F5 B5 2D E8 32 D3
		2-2	X	3C 7E 2C AB FD 7F 04 57 50 F8 4C 5B 22 7A 6A 43
			Y	2D FF 9B 33 51 49 5C E0 95 70 BF C1 E1 A7 65 EC
		2-3	X	42 AA 9B 33 51 49 5C E0 95 70 BF C1 E1 A7 65 EC
			Y	B0 7D D3 07 E3 A2 62 CD F0 8B 47 4C 5A 97 04 54
		2-4	X	E1 5f BF B8 63 01 B1 5C 42 6C B6 B7 DF 8C B6 22
			Y	58 03 91 2B 7E C3 0B 9D 36 FA A5 FF 12 91 B1 4D
		2-5	X	3E E8 2E 58 4F 74 82 F2 BE 45 C0 52 48 35 10 6C
			Y	9D 0A 01 C9 25 C6 80 86 76 A5 B5 A5 85 CD D4 65
		2-6	X	37 32 C4 19 38 C6 80 86 76 A5 B5 A5 85 CD D4 65
			Y	82 63 15 29 72 7A BA 94 A8 7B 8E 52 C7 1D 56 E3
		CTR_0		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6c 00 00 00
		S_0		17 32 97 DC CD FF C4 17 5F F2 7B 69 D9 05 BF CE
		T		95 51 82 F5 BF 85 7E 83 F7 89 F5 3B 1E 18 E9 2D
	암호문 계산	CTR_1		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 01
		S_1		08 A5 CC A8 E0 8F 3F 53 31 2F 3E A0 9F 78 FE 12
		C_1		59 87 A0 17 60 2C EC C2 83 C8 CF 5B 1A 63 4C 64
		CTR_2		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 02
		S_2		7E 3C 39 51 90 55 4D 6F BC 93 EA AB 91 27 D1 AD
		C_2		18 D7 86 22 A1 E2 C4 00 34 2C 8F 06 CB 83 70 8C
		CTR_3		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 03
		S_3		7B 2C 60 0D EF BA B8 34 F0 64 3A 9C 26 0D 5B 3D
		C_3		D1 14 A5 DD F2
출력	암호문(C)			59 87 A0 17 60 2C EC C2 83 C8 CF 5B 1A 63 4C 64 18 D7 86 22 A1 E2 C4 00 34 2C 8F 06 CB 83 70 8C D1 14 A5 DD F2
	MAC값(T)			95 51 82 F5 BF 85 7E 83 F7 89 F5 3B 1E 18 E9 2D

2) 데이터 3 - 복호화

입력	키(Key)			47 86 F7 50 27 CA 81 B9 B7 1C 67 C4 66 81 2B BB
	난스(Nonce)			65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C
	부가 인증 데이터			EB 6A 52 DD 95 81 9C D2 B4 C0 4D C1 01 2C 30 30 B1 4A DC 7F C2 49 BD DD B9 EE 0F 92 58 90 6F 55
	암호문(C)			59 87 A0 17 60 2C EC C2 83 C8 CF 5B 1A 63 4C 64 18 D7 86 22 A1 E2 C4 00 34 2C 8F 06 CB 83 70 8C D1 14 A5 DD F2
	MAC값(T)			95 51 82 F5 BF 85 7E 83 F7 89 F5 3B 1E 18 E9 2D
	MAC 길이			16 byte
처리 과정	B			7A 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 25 00 20 EB 6A 52 DD 95 81 9C D2 B4 C0 4D C1 01 2C 30 30 B1 4A DC 7F C2 49 BD DD B9 EE 0F 92 58 90 6F 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 51 22 6C BF 80 A3 D3 91 B2 E7 F1 FB 85 1B B2 76 66 EB BF 73 31 B7 89 6F 88 BF 65 AD 5A A4 A1 21 AA 38 C5 D0 1D 00 00 00 00 00 00 00 00 00 00 00
		CTR_0		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6c 00 00 00
		S_0		17 32 97 DC CD FF C4 17 5F F2 7B 69 D9 05 BF CE
		T		95 51 82 F5 BF 85 7E 83 F7 89 F5 3B 1E 18 E9 2D
	복호문 계산	CTR_1		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 01
		S_1		08 A5 CC A8 E0 8F 3F 53 31 2F 3E A0 9F 78 FE 12
		P_1		51 22 6C BF 80 A3 D3 91 B2 E7 F1 FB 85 1B B2 76
		CTR_2		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 02
		S_2		7E 3C 39 51 90 55 4D 6F BC 93 EA AB 91 27 D1 AD
		P_2		66 EB BF 73 31 B7 89 6F 88 BF 65 AD 5A A4 A1 21
		CTR_3		02 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 03
		S_3		7B 2C 60 0D EF BA B8 34 F0 64 3A 9C 26 0D 5B 3D
		P_3		AA 38 C5 D0 1D
	MAC값 계산	2-0	B_0	7A 65 C3 DF 96 CE 14 E0 E1 8A 8D FC 6C 00 00 25
			Y	08 92 E8 7E 03 4D 47 52 FF 23 22 10 B2 6F A3 09
		2-1	X	08 B2 03 14 51 90 D2 D3 63 F1 96 D0 FF AE A2 25
			Y	0C 4E 9D E1 21 00 C6 1E ED 25 F5 B5 2D E8 32 D3
		2-2	X	3C 7E 2C AB FD 7F 04 57 50 F8 4C 5B 22 7A 6A 43
			Y	2D FF 9B 33 51 49 5C E0 95 70 BF C1 E1 A7 65 EC
		2-3	X	42 AA 9B 33 51 49 5C E0 95 70 BF C1 E1 A7 65 EC
			Y	B0 7D D3 07 E3 A2 62 CD F0 8B 47 4C 5A 97 04 54
		2-4	X	E1 5f BF B8 63 01 B1 5C 42 6C B6 B7 DF 8C B6 22
			Y	58 03 91 2B 7E C3 0B 9D 36 FA A5 FF 12 91 B1 4D
		2-5	X	3E E8 2E 58 4F 74 82 F2 BE 45 C0 52 48 35 10 6C
			Y	9D 0A 01 C9 25 C6 80 86 76 A5 B5 A5 85 CD D4 65
		2-6	X	37 32 C4 19 38 C6 80 86 76 A5 B5 A5 85 CD D4 65
			Y	82 63 15 29 72 7A BA 94 A8 7B 8E 52 C7 1D 56 E3
출력	평문(P)			51 22 6C BF 80 A3 D3 91 B2 E7 F1 FB 85 1B B2 76 66 EB BF 73 31 B7 89 6F 88 BF 65 AD 5A A4 A1 21 AA 38 C5 D0 1D

마. GCM 모드 참조구현값

1) 데이터 4 - 암호화

입력	키(Key)			D5 9F BC 68 0C 17 42 46 5F 9A 87 53 EA 25 9F 1C
	난스(Nonce)			93 58 BE DC 3E D6 06 FC 40 EF 9C D6 16 7D 01 BE
	부가 인증 데이터			91 AA E5 0D BC 2A 9E 7B E5 93 F6 F5 30 C7 5D FB AD E5 80 58 8A B6 1A A6 87 48 0D 14 A9 D8 B1 67 33 F5 6C 3F D2 E3 83 9C 18 3E 2F A0 2B DF F4 97 BF 04 2E 34 41 AA ED 1D 55 3E 02 AB 83 75 5A EA 2D 79 8B E8 C5 45 AD 2C 3D 52 6B 84 BE 73 59 04 9C FD 89 4B 8A 2C 59 08 0A C2 8E BD 28 F1 A7 C3 68 7B 6B 8F FD 18 C5 33 3B 18 D1 27 4F 49 37 4A 2E 1B B8 24 CB 02 06 6E 8D 1C D9 D4 00 14 3F F9
	평문(P)			58 0B A0 43 E8 3D A3 B2 6F 6C 0B DC 0A 90 00 19 40 63 65 8D 38 C3 B0 AA 28 58 F9 F6 11 78 50 0F AA 47 54 EC 99 23 02 65 E9 3F CA F6 B9 8D E8 C0 31 E2 F1 D0 BB 55 AC A4 6F E8 24 EB CE 6A FB 8D B4 9B 01 EB 88 92 05 68 B6 5C EC 28 5F E6 00 18 4F 1C 8A 2E 2F 53 A2 F2 FB E4 46 3D B7 1A AA 40 5E 4F CF C9 1C 51 57 C2 BC 0B 98 FA 64 61 EF 29 80 5B 56 2E FC 86 39 9A B6 97 86 72 33 52 04 31
	MAC 길이			4 byte
처리 과정	H			63 A5 90 38 9D 04 22 48 E8 21 77 5F 91 D3 0D A9
	CTR_0			F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 77
	암호문 계산	2-1	CTR_1	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 78
			C_1	91 9B 50 2E 4F 21 8D 32 C9 ED 4E 15 16 B3 E3 15
		2-2	CTR_2	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 79
			C_2	48 92 36 5F 5D 77 83 A7 02 59 7E 48 D3 36 A6 75
		2-3	CTR_3	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7a
			C_3	0C C3 5A 37 C5 C6 07 02 C5 FC 5A EF D9 25 DF EB
		2-4	CTR_4	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7b
			C_4	0A DE 29 2C 98 1A 57 09 6B 8B 58 17 E4 4D 57 62
		2-5	CTR_5	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7c
			C_5	0E 54 98 30 F2 FF 72 56 27 80 8C F0 7B A6 03 1C
		2-6	CTR_6	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7d
			C_6	C4 CD 45 CE 4D C8 F0 73 4C 8F B4 29 62 D4 0A 95
		2-7	CTR_7	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7e
			C_7	BC 9F 02 80 FE 43 B4 A6 CC 38 B3 72 A4 09 BF 1C
		2-8	CTR_8	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7f
			C_8	F3 02 8E 10 32 DD 60 3D F8 29 E8 12 F1 A0 DF 0F
	MAC값 계산	S		E9 35 AF 84 C2 21 47 B9 F5 3E 21 E3 20 8B EF 78
		CTR_0		F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 77
		Y		50 F8 AC 67 1B 3B 37 FB 94 D1 16 56 3B BE 46 26
		T		B9 CD 03 E3 D9 1A 70 42 61 EF 37 B5 1B 35 A9 5E
출력	암호문(C)			91 9B 50 2E 4F 21 8D 32 C9 ED 4E 15 16 B3 E3 15 48 92 36 5F 5D 77 83 A7 02 59 7E 48 D3 36 A6 75 0C C3 5A 37 C5 C6 07 02 C5 FC 5A EF D9 25 DF EB 0A DE 29 2C 98 1A 57 09 6B 8B 58 17 E4 4D 57 62 0E 54 98 30 F2 FF 72 56 27 80 8C F0 7B A6 03 1C C4 CD 45 CE 4D C8 F0 73 4C 8F B4 29 62 D4 0A 95 BC 9F 02 80 FE 43 B4 A6 CC 38 B3 72 A4 09 BF 1C F3 02 8E 10 32 DD 60 3D F8 29 E8 12 F1 A0 DF 0F
	MAC값(T)			B9 CD 03 E3

2) 데이터 4 - 암호화

입력	키(Key)			D5 9F BC 68 0C 17 42 46 5F 9A 87 53 EA 25 9F 1C
	난스(Nonce)			93 58 BE DC 3E D6 06 FC 40 EF 9C D6 16 7D 01 BE
	부가 인증 데이터			91 AA E5 0D BC 2A 9E 7B E5 93 F6 F5 30 C7 5D FB AD E5 80 58 8A B6 1A A6 87 48 0D 14 A9 D8 B1 67 33 F5 6C 3F D2 E3 83 9C 18 3E 2F A0 2B DF F4 97 BF 04 2E 34 41 AA ED 1D 55 3E 02 AB 83 75 5A EA 2D 79 8B E8 C5 45 AD 2C 3D 52 6B 84 BE 73 59 04 9C FD 89 4B 8A 2C 59 08 0A C2 8E BD 28 F1 A7 C3 68 7B 6B 8F FD 18 C5 33 3B 18 D1 27 4F 49 37 4A 2E 1B B8 24 CB 02 06 6E 8D 1C D9 D4 00 14 3F F9
	암호문(P)			91 9B 50 2E 4F 21 8D 32 C9 ED 4E 15 16 B3 E3 15 48 92 36 5F 5D 77 83 A7 02 59 7E 48 D3 36 A6 75 0C C3 5A 37 C5 C6 07 02 C5 FC 5A EF D9 25 DF EB 0A DE 29 2C 98 1A 57 09 6B 8B 58 17 E4 4D 57 62 0E 54 98 30 F2 FF 72 56 27 80 8C F0 7B A6 03 1C C4 CD 45 CE 4D C8 F0 73 4C 8F B4 29 62 D4 0A 95 BC 9F 02 80 FE 43 B4 A6 CC 38 B3 72 A4 09 BF 1C F3 02 8E 10 32 DD 60 3D F8 29 E8 12 F1 A0 DF 0F
	MAC값(T)			B9 CD 03 E3
	MAC 길이			4 byte
처리 과정	H			63 A5 90 38 9D 04 22 48 E8 21 77 5F 91 D3 0D A9
	CTR_0			F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 77
	암호문 계산	2-1	CTR_1	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 78
			P_1	58 0B A0 43 E8 3D A3 B2 6F 6C 0B DC 0A 90 00 19
		2-2	CTR_2	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 79
			P_2	40 63 65 8D 38 C3 B0 AA 28 58 F9 F6 11 78 50 0F
		2-3	CTR_3	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7a
			P_3	AA 47 54 EC 99 23 02 65 E9 3F CA F6 B9 8D E8 C0
		2-4	CTR_4	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7b
			P_4	31 E2 F1 D0 BB 55 AC A4 6F E8 24 EB CE 6A FB 8D
		2-5	CTR_5	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7c
			P_5	B4 9B 01 EB 88 92 05 68 B6 5C EC 28 5F E6 00 18
		2-6	CTR_6	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7d
			P_6	4F 1C 8A 2E 2F 53 A2 F2 FB E4 46 3D B7 1A AA 40
		2-7	CTR_7	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7e
			P_7	5E 4F CF C9 1C 51 57 C2 BC 0B 98 FA 64 61 EF 29
		2-8	CTR_8	F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 7f
			P_8	80 5B 56 2E FC 86 39 9A B6 97 86 72 33 52 04 31
	MAC값 계산	S		E9 35 AF 84 C2 21 47 B9 F5 3E 21 E3 20 8B EF 78
		CTR_0		F2 D8 80 67 5D CA 3A D1 04 1D 32 68 C8 90 4A 77
		Y		50 F8 AC 67 1B 3B 37 FB 94 D1 16 56 3B BE 46 26
		T_1		B9 CD 03 E3 D9 1A 70 42 61 EF 37 B5 1B 35 A9 5E
출력	평문(P)			58 0B A0 43 E8 3D A3 B2 6F 6C 0B DC 0A 90 00 19 40 63 65 8D 38 C3 B0 AA 28 58 F9 F6 11 78 50 0F AA 47 54 EC 99 23 02 65 E9 3F CA F6 B9 8D E8 C0 31 E2 F1 D0 BB 55 AC A4 6F E8 24 EB CE 6A FB 8D B4 9B 01 EB 88 92 05 68 B6 5C EC 28 5F E6 00 18 4F 1C 8A 2E 2F 53 A2 F2 FB E4 46 3D B7 1A AA 40 5E 4F CF C9 1C 51 57 C2 BC 0B 98 FA 64 61 EF 29 80 5B 56 2E FC 86 39 9A B6 97 86 72 33 52 04 31

바. CMAC 모드 참조구현값

1) 데이터 5

입력	키(Key)			F9 C5 9D D0 B2 8B B2 9B 74 1B C6 50 BE 41 86 BB
	평문(P)			(NULL)
	MAC 길이			16 byte
처리 과정	보조 비밀키 K1, K2 생성		L	30 E5 B3 20 0A 4C 2F 73 B3 DE 13 D7 E2 9C 88 AA
			K1	61 CB 66 40 14 98 5E E7 67 BC 27 AF C5 39 11 54
			K2	C3 96 CC 80 29 30 BD CE CF 78 4F 5F 8A 72 22 A8
	MAC값 계산	2-0	m	1
			M_m	43 96 CC 80 29 30 BD CE CF 78 4F 5F 8A 72 22 A8
		2-1	X	43 96 CC 80 29 30 BD CE CF 78 4F 5F 8A 72 22 A8
			Y	6A 6F 37 8E CF 4B CB F4 F8 A1 69 13 2E D8 38 13
출력	MAC값(T)			6A 6F 37 8E CF 4B CB F4 F8 A1 69 13 2E D8 38 13