다양한 보안프로토콜에서의 SEED 이용 가이드라인

2008. 6

[목 차]

I. 개 요

1. SEED 소개

SEED는 전자상거래, 금융, 무선통신 등에서 전송되는 중요 정보를 보호하기 위해 1999년 2월 한국인터넷진흥원((구)한국정보보호진흥원)을 중심으로 국내 암호알고리즘 전문가들이 참여하여 순수 국내기술로 개발한 128비트 블록암호알고리즘이다.

개발된 블록암호알고리즘은 정보보호 산업 활성화의 씨앗이 되라는 의미에서 'SEED'라 명명되었으며, 같은 해 9월 한국정보통신기술협회(TTA) 표준으로 제정된 이후 TTA 표준 중 가장 높은 활용도를 보이고 있다.

※ TTA에서 실시한 정보통신표준 활용실태 조사결과에 따르면, SEED는 2000년부터 2003년까지 국내 2,000여 TTA 표준 중, 4년 연속 활용도 1위를 차지함

2. 기술 및 표준화 현황

▣ 표준화 현황

SEED는 1999년 9월 TTA으로 제정되었으며, 2005년에는 국제 표준화 기구인 ISO/IEC 국제 블록암호알고리즘 표준으로 제정되었다. 또한, 같은 해 IETF 표준으로도 제정되었다.

분류	표준명
국내 표준	**TTAS.KO-12.0004/R1 : 128비트 블록암호알고리즘(SEED)**
국제 표준	**ISO/IEC 18033-3 : Information technology - Security techniques - Encryption algorithms - Part 3 : Block ciphers**
	IETF RFC 4269 : The SEED Encryption Algorithm ※ RFC4269는 RFC 4009("The SEED Encryption Algorithm")의 개정 표준임

▣ 관련 표준화 현황

SEED 암호알고리즘 자체에 대한 표준 외에도 보안프로토콜에서 SEED를 사용하기 위한 다양한 국내/외 표준들이 제정되었다.

분류	표준명
국내 표준	**TTAS.KO-12.0025 : 블록암호알고리즘 SEED의 운영모드**
국제 표준	IETF RFC 4010 : 보안전자우편에서의 메시지 암호화를 위한 SEED 사용표준
	IETF RFC 4162 : TLS를 위한 SEED 알고리즘 사용표준
	IETF RFC 4196 : IPsec을 위한 SEED 알고리즘 사용표준

▣ 기타

SEED 암호알고리즘에 대한 소스코드 및 세부 명세서, 알고리즘 참조 구현값, 알고리즘의 안전성 및 효율성에 대한 평가보고서 등이 한국인터넷진흥원 홈페이지(http://www.kisa.or.kr)을 통해 배포하고 있다.

o 배포 자료
- SEED 알고리즘 사양 및 세부 명세
- KISA에서 자체 수행한 SEED 평가보고서 v1.2
- CRYPTREC에 의뢰한 SEED 평가보고서
- 변형된 SEED의 테스트벡터
- 참조구현값
- SEED OID

II. SEED 이용을 위한 가이드라인

본 가이드라인은 국내·외 표준을 기반으로 보안전자우편 및 IPsec, TLS 등 보안 프로토콜에서의 SEED 사용을 위한 방법을 제시한다.

1. 메시지 암호화를 위한 SEED 사용 표준

다음은 보안전자우편과 같은 응용프로그램에서 메시지를 암호화할 때, SEED를 이용하기 위한 규격으로 암호 메시지 규격에서 사용되는 SEED 관련 객체식별자(Object Identifier) 및 처리 방법을 정의한다.

1.1 컨텐츠와 키 암호화를 위한 객체 식별자

본 절에서는 SEED 알고리즘을 암호메시지 규격(CMS, Cryptographic Message Syntax)에서 사용할 수 있는 대칭키 암호알고리즘으로 추가하기 위해 SEED를 이용한 컨텐츠 암호화 알고리즘과 키 암호화 알고리즘에 대한 객체 식별자를 정의한다. 따라서 CMS 에이전트는 대응하는 암호화 알고리즘에 대한 객체 식별자를 선택하고 각각에 요구되는 파라미터를 제공하여 프로그램 코드를 시작함으로써 컨텐츠나 키 암호화를 위해 SEED를 이용할 수 있다.

1.1.1 컨텐트 암호화용 객체 식별자

SEED는 [RFC3370]에서 정의된 대칭 컨텐트 암호화 알고리즘의 부류에 추가된다. Cipher Block Chaining(CBC) 방법에서 SEED 컨텐트-암호화 알고리즘은 다음 객체 식별자를 가지고 있다

```
id-seedCBC OBJECT IDENTIFIER ::=
 { iso(1) member-body(2) korea(410) kisa(200004)
   algorithm(1) seedCBC(4) }
```

CMS에서 컨텐츠 암호화를 위한 대칭키 암호알고리즘으로 SEED를 사용하기 위해서는 SEED CBC에 대한 AlgorithmIdentifier 파라미터 필드는 반드시 존재해야만 하고, 파라미터 필드는 초기값(IV, Initial Value)의 값을 가지고 있어야 한다.

```
SeedCBCParameter ::= SeedIV  --  SEED 초기벡터
SeedIV ::= OCTET STRING (SIZE(16))
```

평문은 [RFC3369]의 6.3장에 따라 덧붙여진다.

1.1.2 키 암호화를 위한 객체 식별자들

SEED 키 암호화 키(KEK : Key-Encryption Key)를 이용하여 SEED 컨텐츠 암호화 키(CEK : Content-Encryption Key)를 암/복호화 하는데 사용되는 키 싸기(Key Wrapping) 및 키 풀기(Key Unwrapping) 알고리즘은 1.2절에서 정의된다. 그러므로 본 절에서는 SEED 키-암호화 알고리즘을 위한 객체식별자만을 정한다. 또한, 키 암호화 키의 생성과 분배는 이 문서의 범위를 넘어서는 것으로, 본 가이드라인에서는 다루지 않는다.

SEED 키-암호화 알고리즘은 다음 객체 식별자를 가지고 있다:

```
id-npki-app-cmsSeed-wrap OBJECT IDENTIFIER ::=
    { iso(1) member-body(2) korea(410) kisa(200004) npki-app(7)
     smime(1) alg(1) cmsSEED-wrap(1) }
```

키를 싸는 과정은 그 자체에서 IV를 사용할 방법과 시점을 정의하기 때문에 이 객체식별자와 관련된 파라미터는 없어야만 한다.

1.2 키 싸기 알고리즘

SEED 키 암호화 키(KEK)를 이용하여 SEED 컨텐츠 암호화 키(CEK)를 암호화 하는데 사용되는 키-싸기 알고리즘은 AES의 키 싸기 알고리즘[RFC3394]와 동일하다.

1.2.1 기호 및 정의

다음 기호법은 키 싸기 알고리즘의 기술에 사용된다.

- o SEED(K, W) : 키 K를 갖고 SEED 코드북(codebook)를 이용해서 W 암호화
- o SEED-1(K, W) : 키 K를 갖고 SEED 코드북를 이용해서 W 복호화
- o MSB(j, W) : W의 최상위 j 비트를 반환

o LSB(j, W) : W의 최하위 j 비트들 반환
o B1 ^ B2 : B1과 B2의 비트연산 XOR(bitwise exclusive or)
o B1 | B2 : B1과 B2의 비트연산 OR(Concatenate)
o K : 키-암호화 키 K
o n : 64 비트 키 데이터 블록의 수
o s : 키 싸기 처리에서 단계의 수, s = 6n
o P[i] : i번째 평문 키 데이터 블럭
o C[i] : i번째 암호문 데이터 블럭
o A : 64비트 무결성 검사 레지스터
o R[i] : 64비트 레지스터 배열 (여기서, i = 0, 1, 2, ..., n)
o A[t], R[i][t] : 암호화 단계 t번 후 레지스터 A와 R[i]의 내용.
o IV : 키 싸기 처리 동안 사용된 64비트 초기값

키 싸기 알고리즘에서, 연접 기능(concatenation function)은 SEED 코드북에 128비트 입력을 구성하기 위하여 64비트의 조각을 연접하기 위하여 사용될 것이다. 또한, 추출 기능은 SEED 코드북으로부터 나온 128 비트 출력을 두개의 64비트 조각으로 나누는데 사용될 것이다.

1.2.2 SEED 키 싸기

SEED 키 싸기 알고리즘은 AES에 대한 키 싸기 알고리즘[RFC3394]의 2.2.1장과 동일하며, 단지 "AES"를 "SEED"로 대치하면 된다.

키 싸는 과정에서의 입력은 키 암호화 키와 싸여진 평문이며, 평문은 싸여진 키 데이터를 포함하는 n 개의 64비트 블록들로 구성된다. 키 싸는 과정은 다음과 같다.

o 입력
- 평문 : n개의 64비트 값 {P1, P2, ..., Pn})
- 키 : K (KEK)

o 출력
- 암호문 : (n+1)개의 64비트 값 {C0, C1, ..., Cn}

o 키 싸기 과정(이동(shifting) 방식)

① 변수 초기화

```
A[0]로 초기값 설정 (4.4장 참조)
For i = 1 to n
    R[0][i] = P[i]
```

② 중간 값 계산

```
For t = 1 to s, where s = 6n
    A[t] = MSB(64, SEED(K, A[t-1] | R[t-1][1])) ^ t
    For i = 1 to n-1
        R[t][i] = R[t-1][i+1]
    R[t][n] = LSB(64, SEED(K, A[t-1] | R[t-1][1]))
```

③ 결과 출력

```
Set C[0] = A[t]
For i = 1 to n
    C[i] = R[t][i]
```

이동(shifting) 방식의 키 싸기 알고리즘이 아닌 색인(indexing) 방식의 키 싸기 알고리즘도 사용할 수 있다. 이 방법을 통해 앞선 알고리즘에 있는 회전을 피하면서 싸여진 키를 계산하는 것이 가능하다. 이 방법은 앞선 방법과 동일한 결과를 얻으면서, 소프트웨어 구현을 보다 쉽게 한다.

o 입력

- 평문 : n개의 64비트 값 {P1, P2, ..., Pn}
- 키 : K (KEK)

o 출력

- 암호문 : (n+1)개의 64비트 값 {C0, C1, ..., Cn}

o 키 싸기 과정(색인(indexing) 방식)

① 변수 초기화

```
A = IV로 초기값 설정 (4.4장 참조)
For i = 1 to n
    R[i] = P[i]
```

② 중간 값 계산

```
        For j = 0 to 5
            For i=1 to n
                B = SEED(K, A | R[i])
                A = MSB(64, B) ^ t where t = (n*j)+i
                R[i] = LSB(64, B)
```

③ 결과 출력

```
        Set C[0] = A
        For i = 1 to n
            C[i] = R[i]
```

1.2.3 SEED 키 풀기

SEED 키 싸기 알고리즘은 AES에 대한 키 싸기 알고리즘[RFC3394]의 2.2.2장과 동일하며, 단지 "AES"를 "SEED"로 대치하면 된다.

키 풀기 과정에 입력되는 값들은 키 암호화 키와 이전에 싸여진 키로 이루어져 있는 암호문인 (n+1)개의 64비트 블록이다. 이것은 키 풀기 과정을 통해 복호화된 키 데이터의 n개 64비트 블록으로 이루어져 있는 n개의 평문 블록을 출력한다.

o 입력
- 암호문 : (n+1)개의 64비트 값 {C0, C1, ..., Cn}
- 키 : K (KEK)

o 출력
- 평문 : n개의 64비트 값 {P1, P2, ..., Pn}.

o 키 풀기 과정(이동(shifting) 방식)

① 변수 초기화

```
        Set A[s] = C[0] where s = 6n
        For i = 1 to n
            R[s][i] = C[i]
```

② 중간 값 계산

```
        For t = s to 1
            A[t-1] = MSB(64, SEED-1(K, ((A[t] ^ t) | R[t][n]))
            R[t-1][1] = LSB(64, SEED-1(K, ((A[t]^t) | R[t][n]))
```

```
            For i = 2 to n
                R[t-1][i] = R[t][i-1]
```

③ 결과 출력

```
        If A[0] is an appropriate initial value (4.4장 참조),
        Then
            For i = 1 to n
                P[i] = R[0][i]
        Else
            Return an error
```

키 풀기 알고리즘은 색인(indexing) 기반의 연산을 사용할 수 있다. 이 방식의 결과는 레지시트 이동(shifting) 방식의 연산과 같은 결과를 생성한다.

o 입력

- 암호문 : (n+1)개의 64비트 값 {C0, C1, ..., Cn}
- 키 : K (KEK)

o 출력

- 평문 : n개의 64비트 값 {P0, P1, K, Pn}.

o 키 풀기 과정(색인(indexing) 방식)

① 변수 초기화.

```
        Set A = C[0]
        For i = 1 to n
            R[i] = C[i]
```

② 중간 값 계산.

```
        For j = 5 to 0
            For i = n to 1
                B = SEED-1(K, (A ^ t) | R[i]) where t = n*j+i
                A = MSB(64, B)
                R[i] = LSB(64, B)
```

③ 결과 출력

```
        If A is an appropriate initial value (4.4장 참조),
        Then
            For i = 1 to n
```

P[i] = R[i]

Else

Return an error

1.2.4 키 데이터 무결성 -- 초기값

초기값(IV, Initial Value)은 키 싸기 과정의 첫번째 단계에서 A[0]에 할당된 값을 참조한다. 이 값은 키 데이터에 대한 무결성 검사를 하는데 사용된다. 키 풀기 과정의 마지막 단계에서 얻은 A[0] 값은 초기의 A[0] 값과 비교된다. 만약 두 값이 일치하면, 키는 유효한 것으로 인정되고 키 풀기 알고리즘은 이 A[0] 값을 반환한다. 만약 일치하지 않으면, 키는 버려지고 키 풀기 알고리즘은 에러를 반환한다.

무결성 검사에 의해 획득된 이러한 엄격한 특징들은 초기값의 정의에 좌우된다. 즉, 다른 애플리케이션은 다소 다른 특성들을 요구할 수도 있다. 예를 들면, 키 데이터의 생명주기(lifecycle) 전체에 걸쳐서 키 데이터의 무결성을 결정한다거나 단순히 키가 풀리는 시점에서 키 데이터의 무결성을 결정할 필요가 있을 수 있다.

키 싸기 과정 동안 키 데이터의 무결성을 제공하는 기본 초기값(default initial value)은 다음과 같은 16진수인 상수로 정의 되어 있다.

A[0] = IV = A6A6A6A6A6A6A6A6

초기값(IV)로서 상수를 사용함으로써 키 데이터를 싸는 동안에 키 데이터에 대한 강한 무결성 검사가 지원된다. 키 데이터를 풀 때 'A[0] = A6A6A6A6A6A6A6A6'로 하면 키 데이터가 훼손된 경우는 2^{-64}이다. 키 데이터를 풀 때 A[0]을 임의의 다른 값으로 하면, 임의의 키 데이터를 되돌리지 않고 오류를 되돌려줘야 한다.

1.3 SMIMECapabilities 속성

S/MIME 클라이언트는 S/MIME 능력 속성(capabilities attribute)을 사용함으로써 지원하는 암호학(cryptographic) 함수의 집합을 발표해야한다[SHOULD]. 이 속성(attribute)은 암호학 함수의 객체 식별자의 부분적인 목록을 제공하고 클라이언트에 의해 서명되어야만 한다. 함수의 객체 식별자는 논리적으로 함수

카테고리에서 분리되어야한다[SHOULD]. 그리고 우선권(preference)에 관하여 정돈되어야만 한다.

[RFC2633]의 2.5.2장에서 SMINMECapabilities 및 이와 관련된 속성을 정의한다(SMIMECapability SEQUENCEs의 SEQUENCE로 정의되었다). 정의된 속성은 SMIMECapabilies들 공시한 소프트웨어가 지원할 수 있는 알고리즘의 일부 항목을 명세 하는데 사용할 수 있다. 만약 S/MIME 클라이언트에게 SEED로 대칭 암호화를 지원하도록 요청되면, 능력 속성은 대칭 알고리즘의 범주 중 위쪽 조건으로 지정된 SEED 객체 식별자를 포함해야만 한다. 이 객체 식별자와 관련된 파라미터는 SeedSMimeCapability이여야만 한다.

SeedSMimeCapabilty ::= NULL

SEED를 기술하는 SMIMECapability SEQUENCE는 다음 16진법 문자열로서 식별 부호화 규칙으로 암호화(DER-encoded)되어야만 한다.

30 0d 06 0a 2a 83 1a 8c 9a 44 07 01 01 01 05 00

송신 에이전트가 암호화된 메시지를 만들 때, 어떤 종류의 암호화 알고리즘을 사용할지를 결정해야만 한다. 일반적으로 결정 과정은 사적인 동의, 사용자 선호, 합법적인 제약 등과 같은 다른 정보뿐만 아니라 수령인으로부터 받은 메시지에서 포함된 성능 목록으로부터 얻는 정보를 포함한다. 만약 사용자들이 대칭 암호화(symmetric encryption)를 위하여 SEED을 요구하면, 송수신 양측에서 S/MIME 클라이언트에 의해 지원되어야만 한다[MUST]. 그리고 사용자가 선호하는 것으로 설정되어야만 한다[MUST].

2. TLS를 위한 SEED 암호알고리즘 사용 표준

다음은 서버와 클라이언트 간의 통신 등에서 보안을 위해 암호통신을 할 때 송수신되는 메시지의 암호화에 SEED를 이용하기 위한 규격으로 CBC 모드에서의 SEED 사용에 대한 식별자와 SEED 키 교환 알고리즘 식별자를 정의한다.

2.1 암호 알고리즘 리스트

현재 TLS에서 지원하고 있는 비대칭 암호 알고리즘(symmetric ciphers)은 RC2, RC4, IDEA, DES, Triple DES, AES 등이 있다[RFC2246, RFC3268]. 본 가이드라인에서는 TLS에서 지원하는 추가 암호 알고리즘으로 SEED를 이용하기 위한 암호 알고리즘 리스트를 정의한다[RFC4269].

SEED 암호 알고리즘의 사용을 위해 새로 정의하는 암호 알고리즘 리스트(Cipher Suites)는 다음과 같다.

암호 알고리즘 리스트
CipherSuite TLS_RSA_WITH_SEED_CBC_SHA = { 0x00, 0x96};
CipherSuite TLS_DH_DSS_WITH_SEED_CBC_SHA = { 0x00, 0x97};
CipherSuite TLS_DH_RSA_WITH_SEED_CBC_SHA = { 0x00, 0x98};
CipherSuite TLS_DHE_DSS_WITH_SEED_CBC_SHA = { 0x00, 0x99};
CipherSuite TLS_DHE_RSA_WITH_SEED_CBC_SHA = { 0x00, 0x9A};
CipherSuite TLS_DH_anon_WITH_SEED_CBC_SHA = { 0x00, 0x9B};

2.2 암호 알고리즘 리스트 정의

2.2.1 암호 알고리즘

SEED 암호 알고리즘 사용을 위해 정의된 모든 암호 알고리즘 리스트는 SEED Cipher Block Chaining (CBC) 운영모드를 이용하고[TTAS/12.0025], 키 및 블록 크기는 128 bits 이다.

2.2.2 해쉬함수

정의되는 모든 암호 알고리즘 리스트는 [RFC2246]에서 정의한 HMAC으로 SHA-1을 이용한다.

2.2.3 키교환 알고리즘 (Key Exchange Algorithms)

정의되는 모든 암호 알고리즘 리스트는 키교환을 위해 다음의 알고리즘을 사용한다.

암호 알고리즘 리스트	키교환 알고리즘
TLS_RSA_WITH_SEED_CBC_SHA	RSA
TLS_DH_DSS_WITH_SEED_CBC_SHA	DH_DSS
TLS_DH_RSA_WITH_SEED_CBC_SHA	DH_RSA
TLS_DHE_DSS_WITH_SEED_CBC_SHA	DHE_DSS
TLS_DHE_RSA_WITH_SEED_CBC_SHA	DHE_RSA
TLS_DH_anon_WITH_SEED_CBC_SHA	DH_anon

※ 참고 : 키교환 알고리즘에 따른 인증서 키 타입

본 규격에서 사용되는 키교환 알고리즘 RSA, DH_DSS, DH_RSA, DHE_DSS, DHE_RSA 및 DH_anon에 사용되는 인증서 키 타입은 다음을 참조한다.

키교환 알고리즘	인증서 키 타입
RSA	RSA public key; the certificate must allow the key to the used for encryption
DH_DSS	Diffie-Hellman key. The algorithm used to sign the certifiate should be DSS(Digital Signature Standard)
DH_RSA	Diffie-Hellman key. The algorithm used to sign the certificate should be RSA
DHE_DSS	DSS public key
DHE_RSA	RSA public key which can be used for signing
DH_anon	anonymous Diffie-Hellman communications

3. IPsec을 위한 SEED 암호알고리즘 사용 표준

다음은 가상사설망, 침입차단시스템 등에서 송수신되는 메시지의 암호화에 SEED를 이용하기 위한 규격으로 IPsec의 ESP(Encapsulating Security Payload) 프로토콜에서 SEED 사용을 위한 요구사항 및 protocol)에서 SEED를 사용하기 위해 관련 객체식별자를 정의한다.

3.1 ESP Payload

SEED는 송신자의 인증 및 데이터 암호화를 함께 지원하는 ESP 프로토콜에서 데이터 암호화를 위해 사용된다. SEED로 암호화된 데이터를 포함하는 ESP Payload는 128 bits(16 octets)의 초기벡터와 암호화된 payload 데이터로 구성된다. 이때 암호화된 payload는 128 bits의 배수이다.

초기벡터 (16 옥텟)
~ 암호화된 payload (가변길이, 16 옥텟의 배수) ~

초기벡터는 임의로 선택되어야 하고, 예측 불가능해야 한다.

3.2 인터넷 키교환에서의 상호작용

인터넷 키교환(IKE, Internet Key Exchange)은 보안연계(SA, Security Association)[1]를 협상하고, ESP 및 AH에서 사용할 키를 관리하기 위한 프로토콜로 안전한 통신 채널 설정을 위한 단계 1과 실제 통신에 사용하게 될 보안연계와 키를 생성하는 단계 2로 구성된다.
다음에서는 인터넷 키교환의 각 단계에서 요구되는 SEED 사용과 관련한 식별자들을 정의한다.

1) 임의의 망 연결이나 연결집합과 관련되는 보안정보 집합. 보안을 위해 두 객체간에 사용되는 암호화 알고리즘 및 키, 보안연계가 사용되는 유효기간 등을 정의

3.2.1 단계 1

인터넷 키교환 프로토콜의 단계 1에서는 보안연계 및 키 관리 프로토콜에 대한 프레임워크인 ISAKMP(Internet Security Association and Key Management Protocol)의 보안연계 협상(negotiation)이 이뤄지는데, SEED를 사용하기 위해서는 SEED CBC 모드에 대한 객체식별자를 다음과 같이 정의한다[RFC4269].

```
algorithm OBJECT IDENTIFIER ::= { iso(1) member-body(2)
                        korea(410) kisa(200004) algorithm(1) }
id-seedCBC OBJECT IDENTIFIER ::= { algorithm seedCBC(4) }
```

3.2.2 단계 2

인터넷 키교환 프로토콜의 단계 2에서는 ESP를 위한 보안연계 협상이 이뤄지는데, SEED를 사용하기 위한 IPsec ESP Transform 식별자를 다음과 같이 정의한다. ESP Transform 식별값은 IANA(Internet Assigned Numbers Authority)로부터 다음과 같이 할당 받았다.

Transform ID	Value
ESP_SEED_CBC	21

3.2.3 키 길이 속성(Key Length Attribute)

SEED 암호알고리즘은 128 bits 키를 가지므로 키 길의 속성은 128 bits로 설정한다.

3.2.4 해쉬 알고리즘

해쉬 알고리즘으로 HMAC-SHA-1과 HMAC-MD5을 사용할 수 있다[RFC2403, RFC2404].

4. 블록암호알고리즘 SEED의 운영모드

다음은 가상사설망, 침입차단시스템 등에서 송수신되는 메시지의 암호화에 SEED를 이용하기 위한 규격으로 IPsec의 ESP(Encapsulating Security Payload) 프로토콜에서 SEED 사용을 위한 요구사항 및 protocol)에서 SEED를 사용하기 위해 관련 객체식별자를 정의한다.

※ 사용 기호 및 표기

- o K : 비밀키, P : 평문 데이터, C : 암호문 데이터, IV : 초기값
- o ctr : 초기 카운터 블록
- o Pi : i번째 단계의 평문 블록, Ci : i번째 단계의 암호문 블록
- o Ii : i번째 단계의 SEED의 입력, Oi : i번째 단계의 SEED의 출력
- o EK(·)/DK(·) : 비밀키 K를 이용한 SEED 암호화/복호화 함수
- o X||Y : X와 Y의 연접 연산
- o X⊕Y : X와 Y의 배타적 논리합 연산
- o X mod Y : X를 Y로 나눈 나머지
- o LSBs(X) : X의 하위 s비트, 예) LSB3(10100011) = 011
- o MSBs(X) : X의 상위 s비트, 예) MSB3(10100011) = 101

4.1 ECB 운영모드

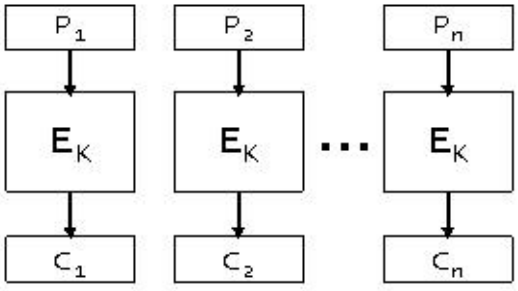
ECB(Electronic Codebook) 모드는 평문 블록을 암호문 블록으로 독립적으로 암호화하는 운영 모드이다.

○ ECB 모드 암호화 의사코드

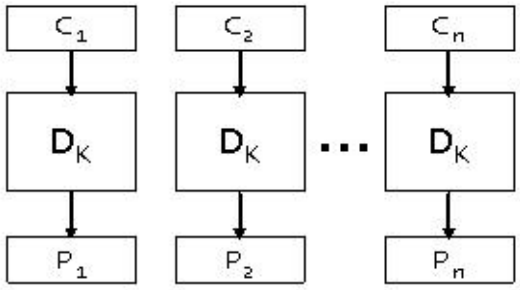
```
o 입력 : P={P1,…,Pn}, K
o 출력 : C={C1,…,Cn}
o 처리과정
   for i=1 to n
       Ii = Pi
       Oi = EK(Ii)
       Ci = Oi
```

○ ECB 모드 복호화 의사코드

```
o 입력 : C={C1,…,Cn}, K
o 출력 : P={P1,…,Pn}
o 처리과정
   for i=1 to n
       Ii = Ci
       Oi = DK(Ii)
       Pi = Oi
```



[ECB 모드 암호화]



[ECB 모드 복호화]

4.2 CBC 운영모드

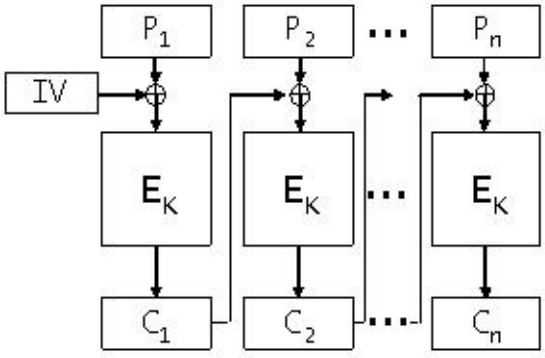
CBC(Cipher Block Chaining) 모드는 동일한 평문 블록과 암호문 블록 쌍이 발생하지 않도록 전 단계의 암·복호화 결과가 현 단계에 영향을 주는 운영 모드이다,

○ CBC 모드 암호화 의사코드

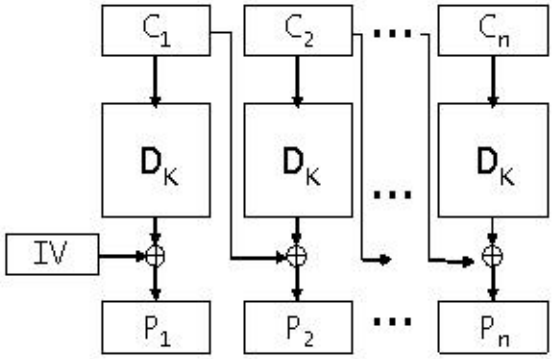
```
o 입력 : P={P1,…,Pn}, K, IV
o 출력 : C={C1,…,Cn}
o 처리과정
    C0 = IV
    for i=1 to n
        Ii = Pi ⊕ Ci-1
        Oi = EK(Ii)
        Ci = Oi
```

○ CBC 모드 복호화 의사코드

```
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 모드 암호화]



[CBC 모드 복호화]

4.3 CFB 운영모드

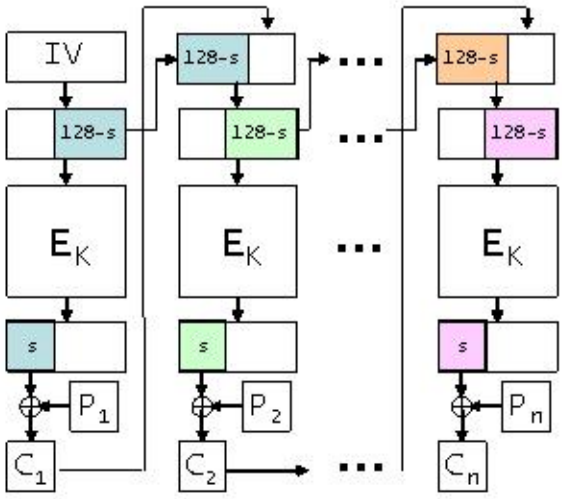
CFB(Cipher FeedBack) 모드에서는 평문 데이터를 sqlxm 평문 블록으로 분할하여 암·복호화를 수행하는 운영모드이다. 단, 본 표준에서는 매개변수 s를 1, 8, 16, 32, 64, 128으로 제한하고, 매개변수 s에 따라 "CFB-s 모드"로 표기한다.

○ CFB 모드 암호화 의사코드

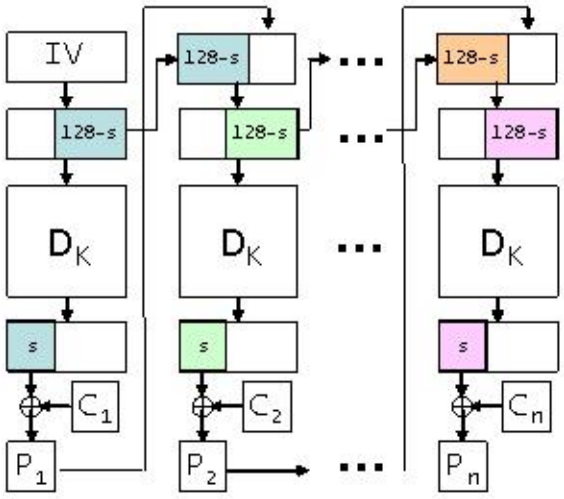
o 입력 : P={P1,…,Pn}, K, IV, s
o 출력 : C={C1,…,Cn}
o 처리과정
  I1 = IV
  for i=1 to n
    Oi = EK(Ii)
    Ci = Pi ⊕ MSBs(Oi)
    Ii+1 = LSB128-s(Ii) || Ci

○ CFB 모드 복호화 의사코드

o 입력 : C={C1,…,Cn}, K, IV, s
o 출력 : P={P1,…,Pn}
o 처리과정
  I1 = IV
  for i=1 to n
    Oi = DK(Ii)
    Pi = Ci ⊕ MSBs(Oi)
    Ii+1 = LSB128-s(Ii) || Ci



[CFB 모드 암호화]



[CFB 모드 복호화]

4.4 OFB 운영모드

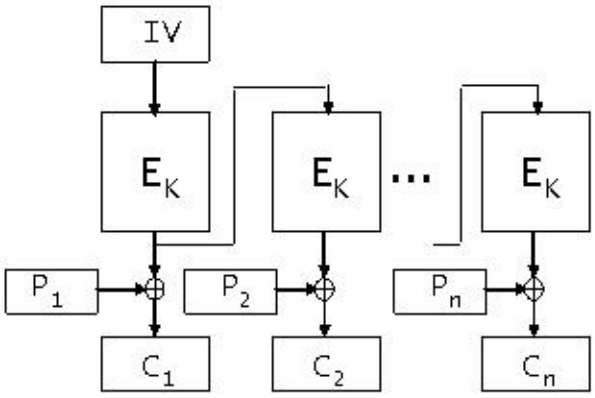
OFB(Output FeedBack) 모드의 전 단계의 출력 블록을 현 단계의 입력 블록으로 사용하는 운영모드이다.

○ OFB 모드 암호화 의사코드

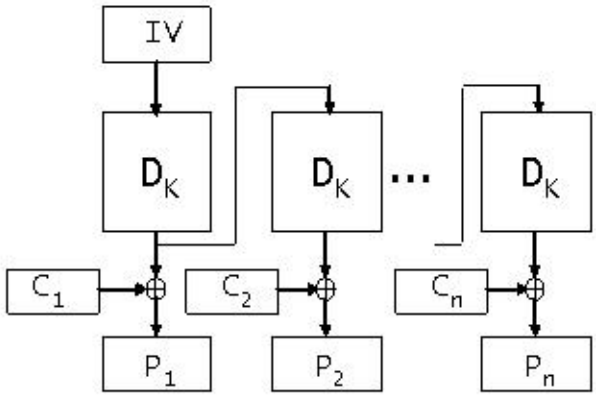
```
o 입력 : P={P1,…,Pn}, K, IV
o 출력 : C={C1,…,Cn}
o 처리과정
   I1 = IV
   for i=1 to n
       Oi = EK(Ii)
       Ci = Pi ⊕ Oi
       Ii+1 = Oi
```

○ OFB 모드 복호화 의사코드

```
o 입력 : C={C1,…,Cn}, K, IV
o 출력 : P={P1,…,Pn}
o 처리과정
   I1 = IV
   for i=1 to n
       Oi = DK(Ii)
       Pi = Ci ⊕ Oi
       Ii+1 = Oi
```



[OFB 모드 암호화]



[OFB 모드 복호화]

4.5 CTR 운영모드

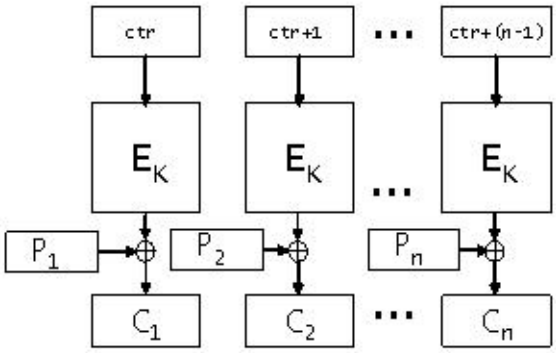
CTR(CounTeR) 모드는 각 단계에 따라 증가되는 카운트 블록을 입력 블록으로 사용하는 운영모드이다.

○ CTR 모드 암호화 의사코드

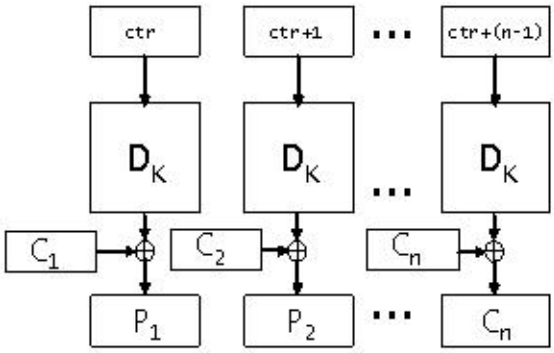
```
o 입력 : P={P1,…,Pn}, K, ctr
o 출력 : C={C1,…,Cn}
o 처리과정
   for i=1 to n
       Ii = ctr+(i-1) mod 2128
       Oi = EK(Ii)
       Ci = Pi ⊕ Oi
```

○ CTR 모드 복호화 의사코드

```
o 입력 : C={C1,…,Cn}, K, ctr
o 출력 : P={P1,…,Pn}
o 처리과정
   for i=1 to n
       Ii = ctr+(i-1) mod 2128
       Oi = DK(Ii)
       Pi = Ci ⊕ Oi
```



[CTR 모드 암호화]



[CTR 모드 암호화]

Ⅲ. 테스트 벡터

1. SEED 암호알고리즘 테스트 벡터

다음은 SEED 암호알고리즘의 구현적합성 검증을 위한 테스트 벡터들로 특정 키와 평문을 입력으로 했을 때 출력되는 암호문 값을 제시한다. 다음의 테스트 벡터 값은 모두 16진수로 표현되어 있다.

o 테스트 벡터 1

키	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
평문	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
암호문	C1 1F 22 F2	01 40 50 50	84 48 35 97	E4 37 0F 43

o 테스트 벡터 2

키	47 06 48 08	51 E6 1B E8	5D 74 BF B3	FD 95 61 85
평문	83 A2 F8 A3	99 64 1F B9	A4 E9 A5 CC	2F 13 1C 7D
암호문	EE 54 D1 3E	BC AE 70 6D	22 6B C3 14	2C D4 0D 4A

o 테스트 벡터 3

키	28 DB C3 BC	49 FF D8 7D	CF A5 09 B1	1D 42 2B E7
평문	B4 1E 6B E2	EB A8 4A 14	8E 2E ED 84	59 3C 5E C7
암호문	9B 9B 7B FC	D1 81 3C B9	5D 0B 36 18	F4 0F 51 22

2. IPsec에서의 SEED 암호알고리즘 테스트 벡터

다음은 IPsec에서 데이터 암호화에 SEED CBC 모드들 이용한 테스트 벡터로, 트랜스포트 모드와 터널 모드 각각에 대한 테스트 벡터를 제공한다.

2.1 트랜스포트 모드에서의 ESP 패킷 (ping 192.168.123.100)

- Key : 90D382B4 10EEBA7A D938C46C EC1A82BF
- SPI : 4321
- Source address : 192.168.123.3
- Destination address : 192.168.123.100
- Sequence number : 1
- IV : E96E8C08 AB465763 FD098D45 DD3FF893

< Original packet >

- IP header(20 bytes) : 45000054 08F20000 4001F9FE C0A87B03 C0A87B64
- Data (64 bytes) :

08000EBD	A70A0000	8E9C083D	B95B0700
08090A0B	0C0D0E0F	10111213	14151617
18191A1B	1C1D1E1F	20212223	24252627
28292A2B	2C2D2E2F	30313233	34353637

< Augment data with >

- Padding : 01020304 05060708 090A0B0C 0D0E
- Pad length : 0E
- Next header : 01 (ICMP)

< Pre-encryption Data with padding, pad length and next header(80 bytes) >

08000EBD	A70A0000	8E9C083D	B95B0700
08090A0B	0C0D0E0F	10111213	14151617
18191A1B	1C1D1E1F	20212223	24252627
28292A2B	2C2D2E2F	30313233	34353637
01020304	05060708	090A0B0C	0D0E0E01

< Post-encryption packet with SPI, Sequence number, IV >

- IP Header : 45000054 08F20000 4001F9FE C0A87B03 C0A87B64
- SPI/Seq # : 00004321 00000001
- IV : E96E8C08 AB465763 FD098D45 DD3FF893
- Encrypted Data (80 bytes) :

E7EBAA03	CF45EF09	021B3011	B40D3769
BE96EBAE	CD4222F6	B6F84CE5	B2D5CDD1
60EB6B0E	5A47D16A	501A4D10	7B2D7CC8
AB86BA03	9A000972	66374FA8	F87EE0FB
EF3805DB	FAA144A2	334A34DB	0B0F81CA

2.2 트랜스포트 모드에서의 ESP 패킷 (ping -p 77 -s 20 192.168.123.100)

- Key : 90D382B4 10EEBA7A D938C46C EC1A82BF
- SPI : 4321
- Source address : 192.168.123.3
- Destination address : 192.168.123.100
- Sequence number : 8
- IV : 69D08DF7 D203329D B093FC49 24E5BD80

< Original packet >

- IP header(20 bytes) : 45000030 08FE0000 4001FA16 C0A87B03 C0A87B64
- Data (28 bytes) :

0800B5E8	A80A0500	A69C083D	0B660E00
77777777	*77777777*	*77777777*	

< Augment data with >

- Padding : 0102
- Pad length : 02
- Next header : 01 (ICMP)

< Pre-encryption Data with padding, pad length and next header(32 bytes) >

0800B5E8	A80A0500	A69C083D	0B660E00
77777777	*77777777*	*77777777*	01020201

< Post-encryption packet with SPI, Sequence number, IV >

- IP header : 4500004C 08FE0000 4032F9C9 C0A87B03 C0A87B64
- SPI/Seq # : 00004321 00000008
- IV : 69D08DF7 D203329D B093FC49 24E5BD80
- Encrypted Data (32 bytes) :

B9AD6E19	E9A6A2FA	02569160	2C0AF541
DB0B0807	E1F660C7	3AE2700B	5BB5EFD1

2.3 터널 모드에서의 ESP 패킷 (ping 192.168.123.200)

- Key : 01234567 89ABCDEF 01234567 89ABCDEF
- SPI : 8765
- Source address : 192.168.123.3
- Destination address : 192.168.123.200
- Sequence number : 2
- IV : F4E76524 4F6407AD F13DC138 0F673F37

< Original packet >

- IP header(20 bytes) : 45000054 09040000 4001F988 C0A87B03 C0A87BC8
- Data (64 bytes) :

08009F76	A90A0100	B49C083D	02A20400
08090A0B	0C0D0E0F	10111213	14151617
18191A1B	1C1D1E1F	20212223	24252627
28292A2B	2C2D2E2F	30313233	34353637

< Augment data with >

- Padding : 01020304 05060708 090A
- Pad length : 0A
- Next header : 04 (IP-in-IP)

< Pre-encryption Data with original IP header, padding, pad length and next header (96 bytes) >

45000054	09040000	4001F988	C0A87B03
C0A87BC8	08009F76	A90A0100	B49C083D
02A20400	08090A0B	0C0D0E0F	10111213
14151617	18191A1B	1C1D1E1F	20212223
24252627	28292A2B	2C2D2E2F	30313233
34353637	01020304	05060708	090A0A04

< Post-encryption packet with SPI, Sequence number, IV >

- IP header : 4500008C 09050000 4032F91E C0A87B03 C0A87BC8
- SPI/Seq # : 00008765 00000002
- IV : F4E76524 4F6407AD F13DC138 0F673F37
- Encrypted Data (96 bytes):

2638AA7B	05E71B54	9348082B	67B47B26
C565AED4	737F0BCB	439C0F00	73E7913C
3C8A3E4F	5F7A5062	003B78ED	7CA54A08
C7CE047D	5BEC14E4	8CBA1005	32A12097
8D7F5503	204EF661	729B4EA1	AE6A9178
59A5CAAC	46E810BD	7875BD13	D6F57B3D

2.4 터널모드에서의 ESP 패킷 (ping -p ff -s 40 192.168.123.200)

- Key : 01234567 89ABCDEF 01234567 89ABCDEF
- SPI : 8765
- Source address : 192.168.123.3
- Destination address : 192.168.123.200
- Sequence number : 5
- IV : 85D47224 B5F3DD5D 2101D4EA 8DFFAB22

< Original packet >

- IP header(20 bytes) : 45000044 090C0000 4001F990 C0A87B03 C0A87BC8

· Data (48 bytes) :

0800D63C AA0A0200 C69C083D A3DE0300
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

< Augment data with >

· Padding : 01020304 05060708 090A
· Pad length : 0A
· Next header : 04 (IP-in-IP)

< Pre-encryption Data with original IP header, padding, pad length and next header (80 bytes) >

45000054 09040000 4001f988 C0A87B03
C0A87BC8 0800D63C AA0A0200 C69C083D
A3DE0300 FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF 01020304 05060708 090a0a04

< Post-encryption packet with SPI, Sequence number, IV >

· IP header : 4500007C 090D0000 4032F926 C0A87B03 C0A87BC8
· SPI/Seq # : 00008765 00000005
· IV : 85D47224 B5F3DD5D 2101D4EA 8DFFAB22
· Encrypted Data (80 bytes) :

311168E0 BC36AC4E 59802BD5 192C5734
8F3D29C8 90BAB276 E9DB4702 91F79AC7
79571929 C170F902 FFB2F08B D448F782
31671414 FF29B7E0 168E1C87 09BA2B67
A56E0FBC 4FF6A936 D859ED57 6C16EF1B

3. SEED 운영모드별 테스트 벡터

다음은 SEED 운영모드 표준의 구현적합성 검증을 위한 테스트 벡터이다. 테스트 벡터를 생성하기 위한 평문 데이터와 키, 초기값, 초기 카운트는 아래 표와 같다. 테스트 벡터는 512 및 1,280 비트 평문 데이터에 대한 암호문 데이터를 제공한다.

CFB-1 모드의 512비트 데이터에 대한 테스트 벡터에서 평문 블록과 암호문 블록이 1비트를 나타내는 것을 제외하면 모든 테스트 벡터는 16진법으로 표기된다.

다음은 512 및 1,280 비트 평문 데이터와 키, 초기값, 초기 카운터 값이다.

o 512 비트

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

o 1,280 비트

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

3.1 ECB 모드

o 512 비트 평문에 대한 테스트 벡터

암호문(C)	블록1	0F 4E 7F C7 8C 48 D5 AD 95 10 BA D8 98 7B A5 22
	블록2	39 86 9D 94 74 48 58 38 24 99 67 68 D1 31 F0 A5
	블록3	D6 8C 46 19 B7 C3 45 A7 80 CB 8E 16 77 0B 25 9A
	블록4	36 8C B3 C5 B5 B1 2F FD 78 08 6F D0 5F 39 FC DC

o 1,280 비트 평문에 대한 테스트 벡터

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

3.2 CBC 모드

o 512 비트 평문에 대한 테스트 벡터

암호문(C)	블록1	A2 93 EA E9 D9 AE BF AC 37 BA 71 4B D7 74 E4 27
	블록2	E8 B7 06 D7 E7 D9 A0 97 22 86 39 E0 B6 2B 3B 34
	블록3	CE D1 16 09 CE F2 AB AA EC 2E DF 97 93 08 F3 79
	블록4	C3 15 27 A8 26 77 83 E5 CB A3 53 89 82 B4 8D 06

o 1,280 비트 평문에 대한 테스트 벡터

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

3.3 CFB-128 모드

o 512 비트 평문에 대한 테스트 벡터

암호문(C)	블록1	A3 60 7C D8 82 0D A1 4B 07 87 53 7F AD 11 CA FD
	블록2	0E 6E 08 B6 F1 AA 5E 23 35 01 BE C7 C7 11 58 2C
	블록3	75 51 5A 05 FC 1E D1 27 ED C4 24 1E B7 39 44 CC
	블록4	80 02 6D CC 49 10 73 81 75 F0 4A 85 EA D1 ED A5

o 1,280 비트 평문에 대한 테스트 벡터

암호문(C)	블록1	76 A7 E8 49 DB F8 BD 17 7D 55 ED 16 26 BE 31 78
	블록2	14 40 1A 9D C5 91 F7 9E D1 C7 94 AC 24 20 0B 52
	블록3	B4 7A AE 33 43 93 75 BA FD C2 82 72 51 6F DF A4
	블록4	51 68 CC 4C 75 45 7A 93 B9 F9 82 01 A3 E7 EC 22
	블록5	FA A3 70 E1 26 AA ED 25 CC 17 C3 4D 16 6B 23 BC
	블록6	A3 DC B7 70 71 1B C2 83 7F 1A 8B 67 EF A0 84 E1
	블록7	B8 C9 DB 2D 73 21 C2 25 92 67 39 D5 E9 0B 55 6C
	블록8	C2 33 F0 4B D0 9B 1C 1A EA E1 3B 62 56 14 E9 73
	블록9	E1 5A 91 70 35 2B 74 3D 7B 33 05 A5 13 62 20 E5
	블록10	8A CE 14 D0 FE 4F 6F B0 15 DD 5F 9A EE AF 6E CC

3.4 OFB 모드

o 512 비트 평문에 대한 테스트 벡터

암호문(C)	블록1	A3 60 7C D8 82 0D A1 4B 07 87 53 7F AD 11 CA FD
	블록2	9F EB EC E0 C7 09 96 5B 21 05 18 FB D1 1C 0F 86
	블록3	17 AB 3D 86 E4 20 C5 FD D6 F7 7F 96 9B 40 6D 78
	블록4	E0 53 D1 87 4C 09 EB E9 8A DA 7D 5F 4B 0F 96 4A

o 1,280 비트 평문에 대한 테스트 벡터

암호문(C)	블록1	76 A7 E8 49 DB F8 BD 17 7D 55 ED 16 26 BE 31 78
	블록2	99 B0 FA 58 42 63 5F BC 81 D9 2C 89 86 05 A0 CC
	블록3	1C FC C6 C8 70 2F 6B 5F 91 DF 2F AD 38 25 DA 6E
	블록4	D2 83 66 AF 30 DC F2 4B 54 7A 32 7A A9 91 42 3B
	블록5	B2 AC EC C7 73 2C D6 4C A3 10 2A B3 CB 66 F7 BF
	블록6	C5 BD 23 F2 41 CF 6E 28 67 2C 13 03 58 04 4E CC
	블록7	B2 55 51 A2 10 24 80 D4 DC 34 D8 F6 51 0C 48 C5
	블록8	A4 2A 76 73 85 32 18 6A FB 38 D8 AF D5 21 D1 D9
	블록9	34 D2 52 A2 45 8D AA 05 06 2A E8 C5 0C 10 44 76
	블록10	36 A1 E6 A4 E8 B1 F7 C7 F7 5A 4C 1C 09 03 AA 12

3.5 CTR 모드

o 512 비트 평문에 대한 테스트 벡터

암호문(C)	블록1	54 1E 1C C4 57 A6 08 3E E9 FB 8A 9C 32 27 41 ED
	블록2	94 3A 2C ED 25 5A 9C 30 F8 D1 3E 10 32 8A F5 45
	블록3	2C 52 DD 82 6B E4 F1 85 16 E7 3D F9 1F CC 1B 5B
	블록4	DB 0D 00 F7 25 24 51 08 23 7C 6B 13 50 E5 F5 05

o 1,280 비트 평문에 대한 테스트 벡터

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

4. 패딩(Padding) 방법

블록암호알고리즘에서 ECB, CBC 모드는 평문 블록을 암호화의 입력으로 사용하기 때문에, 평문 데이터의 크기가 128비트의 양의 정수배가 되도록 패딩 방법이 적용되어야만 한다.

본 가이드라인에서 소개하는 패딩 방법은 ISO/IEC 국제표준 및 PKCS에서 사용되는 방법으로, 적용되는 시스템에 맞는 방법을 선택하여 사용함을 권고한다. 단, '패딩 방법 1'의 경우, 복호화할 평문 데이터와 추가로 패딩된 값과의 구분이 모호할 수 있으므로 평문 데이터의 크기가 명확히 알려져 있는 경우에 사용함을 권고한다. 패딩 방법의 사용 예는 16진법 바이트 단위로 표기된다.

4.1 패딩 방법 1

평문 데이터의 크기가 $(128\times t+m)$비트$(0\le m<128,\ 0\le t)$일 때, m이 0이 아닐 경우, 평문 데이터의 크기가 128비트의 양의 정수배가 되도록 평문 데이터의 끝에 (128-m)개의 '0' 비트를 패딩한다.

예) 평문 데이터(48bits) : 4F 52 49 54 48 4D
적용 결과(128bits) : 4F 52 49 54 48 4D 00 00 00 00 00 00 00 00 00 00

예) 평문 데이터(48bits) : 53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78
적용 결과(128bits) : 53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78

4.2 패딩 방법 2

평문 데이터의 크기가 $(128\times t+m)$비트$(0\le m<128,\ 0\le t)$일 때, m이 0이 아닐 경우, 평문 데이터의 크기가 128비트의 양의 정수배가 되도록 평문 데이터의 끝에 비트 '1'을 추가한 후 (128-m-1)개의 '0' 비트를 패딩한다. 또한, m이 0인 경우에는 덧붙이기 방법이 사용됨을 표기하기 위해, 추가적인 128비트 '10…00' 블록을 추가한다.

예) 평문 데이터(48bits) : 4F 52 49 54 48 4D
적용 결과(128bits) : 4F 52 49 54 48 4D 80 00 00 00 00 00 00 00 00 00

예) 평문 데이터(48bits) : 53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78

적용 결과(128bits) : 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

4.3 패딩 방법 3

본 패딩 방법은 바이트 단위로만 적용이 되는 방법이다. 평문 데이터의 크기가 (16×t+m)바이트(0≤m<16, 0≤t)일 때, m이 0이 아닐 경우 평문 데이터의 크기가 16바이트의 양의 정수배가 되도록 평문 데이터의 끝에 덧붙이기가 필요한 바이트 수 (16-m)을 덧붙인다. m이 0인 경우에는 덧붙이기 방법이 사용됨을 표기하기 위해, 추가적인 16바이트 '10…10' 블록을 추가한다.

예) 평문 데이터(6bytes): 4F 52 49 54 48 4D
적용 결과(16bytes) : 4F 52 49 54 48 4D 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A

예) 평문 데이터(6bytes): 53 45 45 44 41 4C 47 A8 3E D1 AF 07 4A 73 12 2C
적용 결과(32bytes) : 53 45 45 44 41 4C 47 A8 3E D1 AF 07 4A 73 12 2C
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10