블록암호 LEA 소스코드
사용 매뉴얼

ver 1.0

2015년 10월

NSR 국가보안기술연구소

<차례>

1. 블록암호 LEA

LEA(Lightweight Encryption Algorithm)는 다음의 입출력 규격을 가지는 블록암호이다.

- 키 길이: 128비트, 192비트, 또는 256비트
- 블록 길이: 128비트

LEA의 세부 규격은 WISA 2013에서 발표되었으며 TTA 표준(TTA-KO-12.0223)으로 제정되었다. LEA는 키 길이에 따라 LEA-128, LEA-192, LEA-256으로 표기한다.

2. 운영모드

2.1. 운영모드 개요

블록암호 자체만으로는 정해진 입력 길이 (블록 길이, LEA의 경우 128비트 = 16바이트)의 데이터에 대한 암호화 및 복호화만 가능하다 임의 입력 길이의 데이터를 암호화 및 복호화하기 위해서는 운영모드(mode of operation)를 적용해야 한다. 블록암호 운영모드에 따라서 데이터 암호화(복호화) 뿐만 아니라, 메시지 인증 또는 메시지 인증과 암호화(복호화)의 기능을 동시에 제공하기도 한다. 운영모드들은 각기 다른 안전성과 특성을 가지고 있기 때문에, 환경에 따라서 적절한 운영모드를 적용해야 한다. 본 문서에서는 다음의 운영모드들을 다룬다.

- 암호화 운영모드: ECB, CBC, CTR, CFB, OFB
- 암호화 및 메시지 인증용 운영모드(인증 암호화 운영모드): CCM, GCM
- 메시지 인증용 운영모드(메시지 인증 코드): CMAC

운영모드는 기본적으로 암호화 시 암호키(128, 192, 또는 256비트)와 평문 바이트열을 입력으로 받는다. 그 이외에 운영모드에 따라 다음과 같은 바이트열을 추가 입력으로 받는다.

- 초기화 벡터(IV): CBC, CFB, OFB
- 초기 카운터: CTR
- Nonce: CCM, GCM
- 부가 인증 데이터(AAD): CCM, GCM

LEA 운영모드에 대한 상세 규격은 TTA 표준(TTAK.KO-12.0246)으로 제정되었다.

2.2. 운영모드별 입출력 규격

본 절에서는 LEA의 각 운영모드별 입출력 규격을 제시한다.

2.2.1. ECB(Electronic Codebook) 모드

ECB 모드 암호화는 암호키와 평문 바이트열을 입력으로 가지고, 암호문 바이트열을 출력으로 가진다. ECB 모

드 암호화를 적용하기 위해서는 우선 사전에 정해진 패딩 방식을 적용하여 평문의 길이가 16바이트의 배수가 되도록 만들어야 한다. ECB 모드에서 암호문의 길이는 16바이트의 배수이다. ECB 모드 복호화는 암호키와 암호문 바이트열을 입력으로 가지고, 평문 바이트열을 출력으로 가진다. 복호화에 사용되는 암호키가 암호화에 사용된 값과 동일해야 평문이 올바르게 복구된다.

2.2.2. CBC(Cipher Block Chaining) 모드

CBC 모드 암호화는 암호키, 평문 바이트열 및 16바이트 IV를 입력으로, 암호문 바이트열을 출력으로 가진다. CBC 모드 암호화를 적용하기 위해서는 우선 사전에 정해진 패딩 방식을 적용하여 평문의 길이가 16바이트의 배수가 되도록 만들어야 한다. CBC 모드에서 암호문의 길이는 16바이트의 배수이다. CBC 모드 복호화는 암호키, 암호문 바이트열과 및 16바이트 IV를 입력으로 가지고, 평문 바이트열을 출력으로 가진다. 복호화에 사용되는 암호키와 IV가 암호화에 사용된 값들과 동일해야 평문이 올바르게 복구된다.

2.2.3. CTR(Counter) 모드

CTR 모드 암호화는 암호키, 평문 바이트열 및 16바이트 초기 카운터를 입력으로, 암호문 바이트열을 출력으로 가진다. CTR 모드에서 평문은 임의의 길이를 가지며, 암호문의 길이는 평문의 길이와 같다. CTR 모드 복호화는 암호키, 암호문 바이트열 및 16바이트 초기 카운터를 입력으로 가지고, 평문 바이트열을 출력으로 가진다. 복호화에 사용되는 암호키와 초기 카운터가 암호화에 사용된 값들과 동일해야 평문이 올바르게 복구된다.

2.2.4. CFB(Cipher Feedback) 모드

CFB 모드 암호화는 암호키, 평문 바이트열 및 16바이트 IV를 입력으로, 암호문 바이트열을 출력으로 가진다. CFB 모드에서 평문은 임의의 길이를 가지며, 암호문의 길이는 평문의 길이와 같다. CFB 모드 복호화는 암호키, 암호문 바이트열 및 16바이트 IV를 입력으로 가지고, 평문 바이트열을 출력으로 가진다. 복호화에 사용되는 암호키와 IV가 암호화에 사용된 값들과 동일해야 평문이 올바르게 복구된다. CFB 모드는 블록암호 1회 동작 시 출력하는 비트 수가 r일 경우 CFB-r로 표기하는데, 본 문서에서 CFB는 CFB-128을 의미한다.

2.2.5. OFB(Output Feedback) 모드

OFB 모드 암호화는 암호키, 평문 바이트열 및 16바이트 IV를 입력으로, 암호문 바이트열을 출력으로 가진다. OFB 모드에서 평문은 임의의 길이를 가지며, 암호문의 길이는 평문의 길이와 같다. OFB 모드 복호화는 암호키, 암호문 바이트열 및 16바이트 IV를 입력으로 가지고, 평문 바이트열을 출력으로 가진다. 복호화에 사용되는 암호키와 IV가 암호화에 사용된 값들과 동일해야 평문이 올바르게 복구된다.

2.2.6. CCM(Counter with CBC-MAC) 모드

CCM 모드 암호화는 암호키, 평문 바이트열, Nonce 및 부가 인증 데이터(AAD)를 입력으로, 암호문 바이트열과 인증값(tag)을 출력으로 가진다. CCM 모드에서 평문은 임의의 길이를 가지며(길이 0도 가능함), Nonce는 7~13바이트이어야 한다. CCM 모드의 인증값의 바이트 길이는 4,6,8,10,12,14,16 중의 하나이어야 한다. CCM 모드 복호화는 암호키, 암호문 바이트열, Nonce, 부가 인증 데이터(AAD) 및 인증값을 입력으로 하며, 인증값이 올바른 경우 평문 바이트열을 출력하고, 올바르지 않은 경우 Invalid를 출력한다. 복호화에 사용되는

암호키, Nonce, AAD가 암호화에 사용된 값들과 동일해야 평문이 올바르게 복구된다.

2.2.7. GCM(Galois/Counter mode) 모드

GCM 모드 암호화는 암호키, 평문 바이트열, Nonce 및 부가 인증 데이터(AAD)를 입력으로 가지고, 암호문 바이트열과 인증값(tag)을 출력으로 가진다. GCM 모드에서 평문은 임의의 길이를 가지며(길이 0도 가능함), Nonce의 길이도 0만 아니면 된다. GCM 모드 복호화는 암호키, 암호문 바이트열, Nonce, 부가 인증 데이터(AAD) 및 인증값을 입력으로 하며, 인증값이 올바른 경우 평문 바이트열을 출력하고, 올바르지 않은 경우 Invalid를 출력한다. 복호화에 사용되는 암호키, Nonce, AAD가 암호화에 사용된 값들과 동일해야 평문이 올바르게 복구된다. GCM 모드 소스코드에서 인증값 바이트 길이는 4 이상이어야 한다.

2.2.8. CMAC

CMAC은 인증값(tag)을 생성 또는 검증하는 기능을 제공한다. CMAC에서 인증값 생성 과정은 암호키와 평문 바이트열을 입력으로, 인증값을 출력으로 가진다. CMAC 평문은 임의의 길이를 가진다(길이 0도 가능함), CMAC에서 인증값 검증 과정은 암호키와 평문 바이트열 및 인증값을 입력으로 하며, 인증값이 올바른지 여부를 출력한다.

3. 소스코드 구성

LEA 소스코드는 다음과 같이 구성되어 있다.

- LEA의 OpenSSL 연동용 C 소스코드
- LEA의 단독 C 소스코드
- LEA의 Java 소스코드
- LEA의 Python 소스코드

각 소스코드 별 적용 환경은 다음 표와 같다.

	OS	CPU	SIMD 가속	비고
OpenSSL 연동용 C 소스코드	Windows	Intel/AMD	적용	MinGW 및 cygwin 환경에서 컴파일 가능
	Linux 등 POSIX 계열	Intel/AMD	적용	크로스 컴파일 지원
		ARM(NEON 지원)	적용	
		기타	미적용	
LEA 단독 C 소스코드	Windows	Intel/AMD	적용	Visual C, MinGW 및 cygwin 환경에서 컴파일 가능
	Linux 등 POSIX 계열	Intel/AMD	적용	크로스 컴파일 지원
		ARM(NEON 지원)	적용	
		기타	미적용	
Java 코드	Java 1.5 이상		미적용	
Python 코드	Python 2.7 이상 또는 3.2 이상		미적용	

OpenSSL 연동용 C 소스코드와 LEA 단독 C 소스코드는 Intel/AMD CPU 및 NEON 지원 ARM 프로세서가 동작하는 환경에서 일부 운영모드의 SIMD 가속을 지원한다. SIMD 가속이 적용되는 운영모드들은 다음과 같다.

- ECB 모드 암호화 및 복호화
- CBC 모드 복호화
- CTR 모드 암호화 및 복호화
- CCM 모드 암호화 및 복호화
- GCM 모드 암호화 및 복호화

CPU와 컴파일러에 따라서 적용되는 SIMD 가속이 다르며, 이와 관련한 세부사항은 4장과 5장에서 다룬다.

4. LEA 단독 C 소스코드

4.1. 서비스

LEA의 C 소스코드는 다음의 서비스를 제공한다.

- LEA의 암호화 운영모드(ECB, CBC, CTR, CFB, OFB)
- LEA의 인증 암호화 운영모드(CCM, GCM)
- LEA 기반의 메시지 인증 코드(CMAC)

4.2. 소스코드의 구성

LEA의 단독 C 소스코드는 다음과 같이 구성되어 있다.

파일명	내용
lea.h	LEA의 인터페이스
lea_locl.h	내부용 헤더 파일
lea_core.c	LEA의 기본 소스코드
lea_online.c	LEA의 온라인 암호화 코드
lea_key.h	LEA의 키스케줄
lea_avx2.h	LEA의 SIMD 템플릿
lea_xop.h	
lea_sse2.h	
lea_neon.h	
lea_ecb.h	운영모드 템플릿
lea_cbc.h	
lea_ctr.h	
lea_cfb.h	
lea_ofb.h	
lea_cmac.h	
lea_ccm.h	
lea_gcm.h	
lea_core_xop.c	LEA 기본 함수의 XOP SIMD 코드
lea_t_generic.c	SIMD별 템플릿 구현체
lea_t_avx2_xop.c	
lea_t_avx2_sse2.c	
lea_t_xop.c	
lea_t_sse2.c	
lea_t_neon.c	

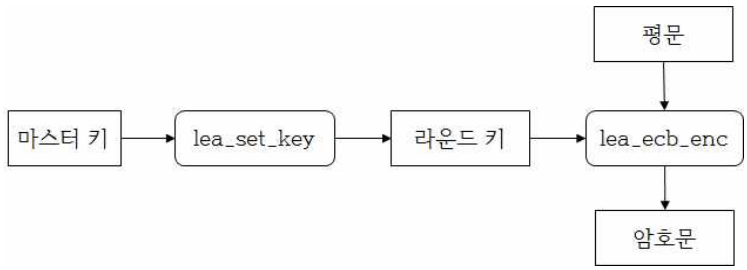
파일명	내용
lea_gcm_generic.c	GCM 모드 소스코드
lea_gcm_pclmul.c	GCM 모드 PCLMUL 적용 소스코드
config.h	수동 설정을 위한 헤더 파일
lea_base.c	SIMD 기능 지원 함수 및 기타 함수
lea_t_fallback.c	
cpu_info_ia32.c	SIMD 지원 여부 확인 코드
cpu_info_arm.c	
arm64cpuid.S	
armv4cpuid.S	
arm_arch.h	
cpu_info.h	

4.3. LEA 단독 C 소스코드의 인터페이스

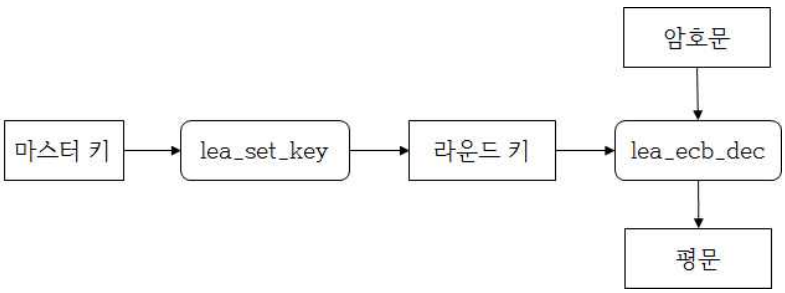
본 절에서는 LEA 단독 C 소스코드가 제공하는 인터페이스를 다룬다.
ECB, CBC, CTR, CFB, OFB 모드를 적용하기 위해서는 먼저 lea_set_key 함수를 수행하여 LEA의 라운드 키들을 생성해야 한다.

ECB 모드

ECB 모드의 암호화와 복호화 과정은 다음과 같이 도시할 수 있다.



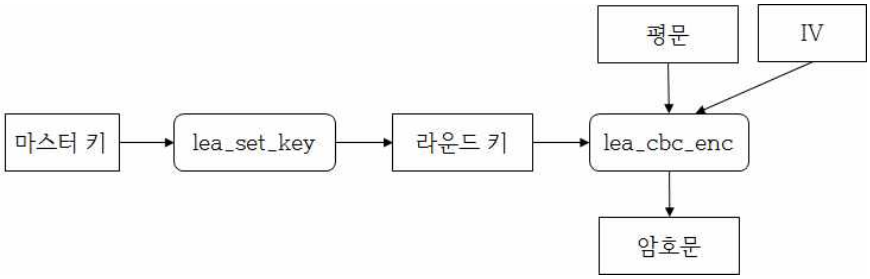
<ECB 모드 암호화 과정>



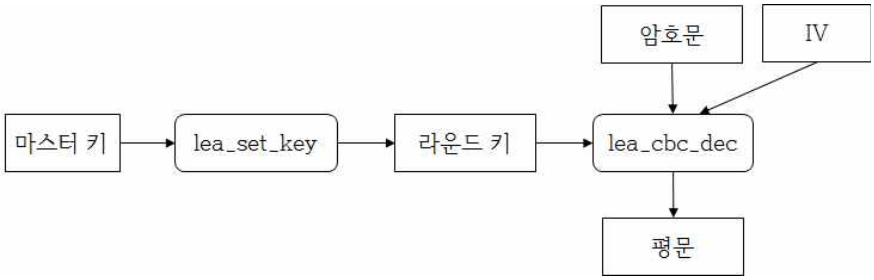
<ECB 모드 복호화 과정>

CBC, CTR, CFB, OFB 모드

CBC 모드의 암호화 및 복호화 과정은 다음과 같이 도시할 수 있다.



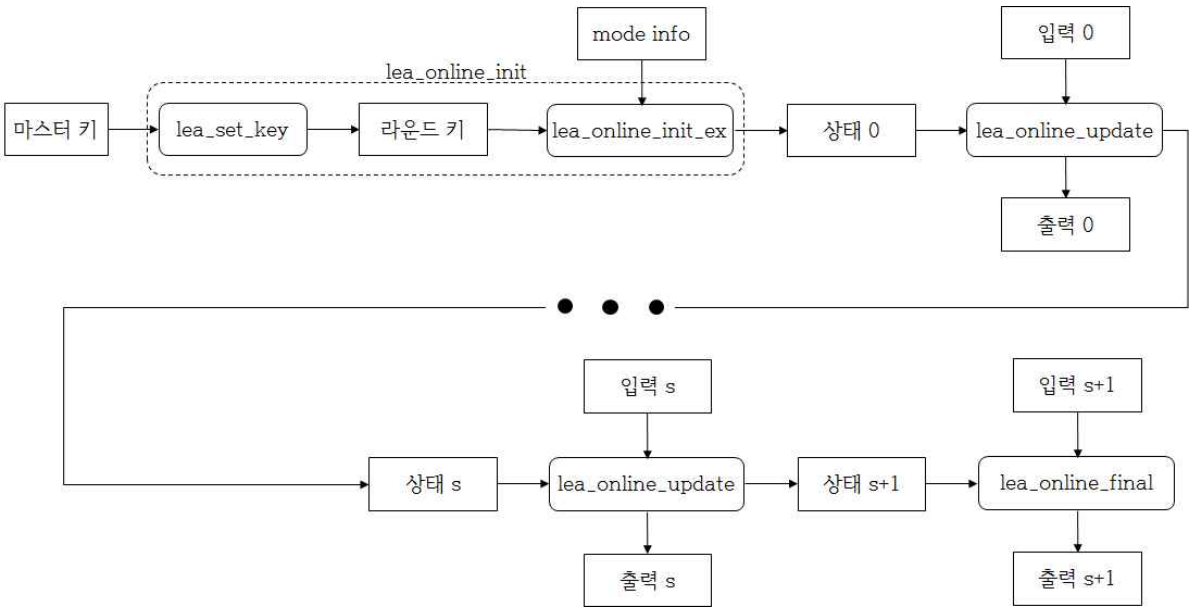
<CBC 모드 암호화 과정>



<CBC 모드 복호화 과정>

CTR, CFB, OFB 모드 암호화 및 복호화 과정은 CBC 모드의 경우와 유사하다.

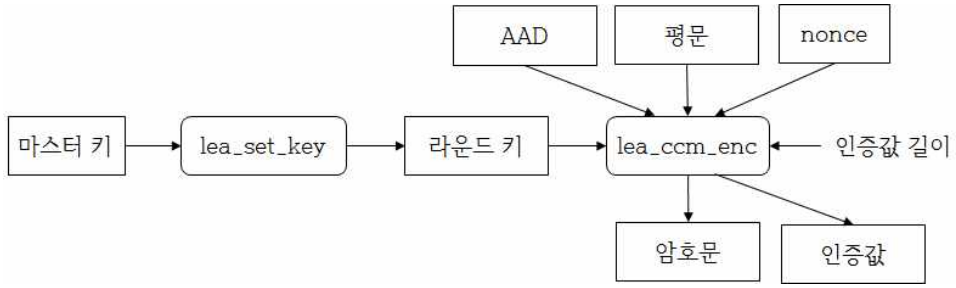
ECB, CBC, CTR, CFB, OFB 모드는 앞에서와 같이 데이터를 한 번에 처리하는 것 뿐만 아니라 입력된 데이터를 나누어 online 처리함으로써, 적용 시스템에서 제공하는 메모리보다 큰 데이터도 처리할 수 있도록 구현되어 있다. ECB 모드와 CBC 모드의 online 처리방식에서 PKCS5 패딩방식을 사용할 수 있으며, 패딩을 사용하지 않는 것으로 설정할 경우, 총 데이터의 길이가 16바이트의 배수이어야 올바르게 동작한다.



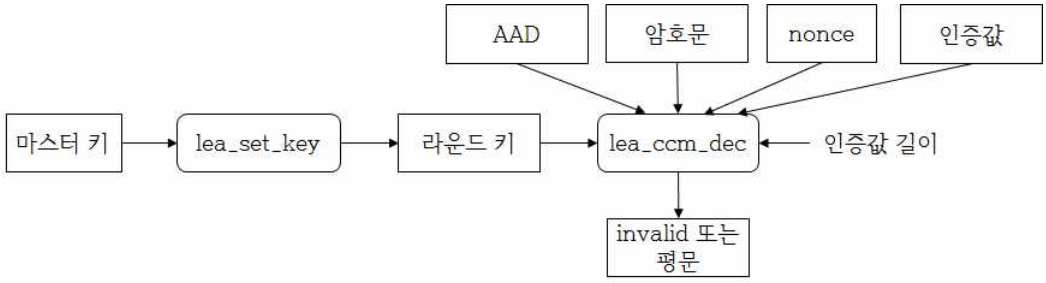
<ECB/CBC/CTR/CFB/OFB 모드 online 암호화 및 복호화 과정>

CCM 모드

CCM 모드를 적용할 경우에도 먼저 lea_set_key 함수를 수행하여 LEA의 라운드 키들을 생성해야 한다. CCM 모드 암호화 및 복호화 과정은 다음과 같이 도시할 수 있다.



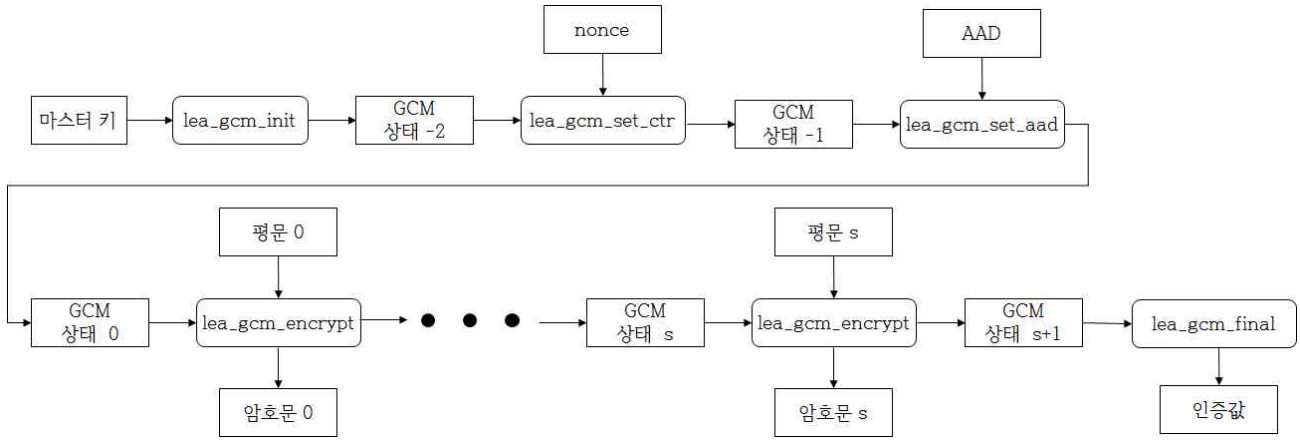
<CCM 모드 암호화 과정>



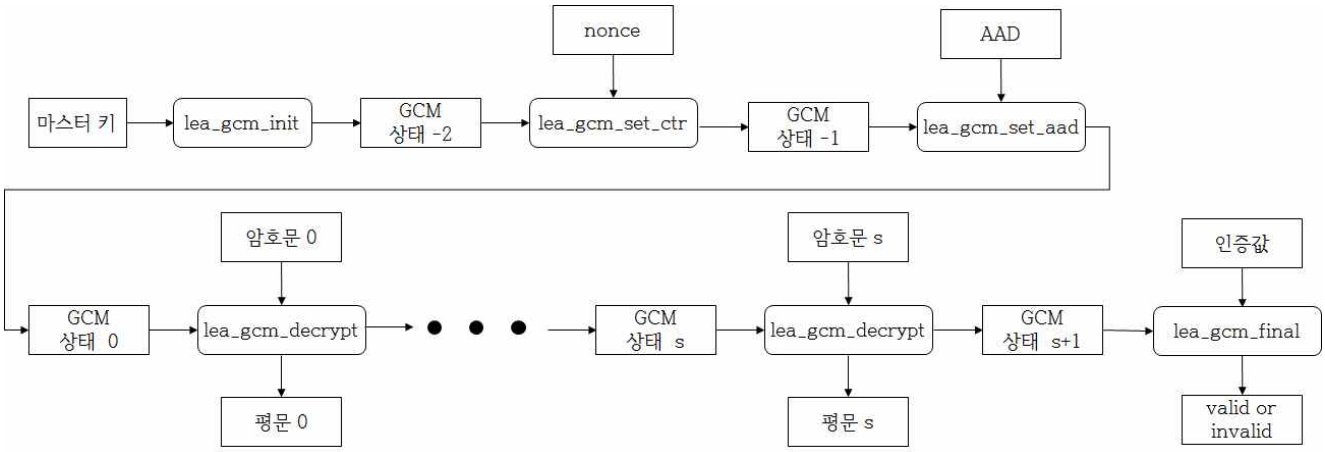
<CCM 모드 복호화 과정>

GCM 모드

GCM 모드는 입력된 데이터를 나누어 online 처리할 수 있도록 구현되어 있다. GCM 모드 암호화는 라운드 키를 생성하고 GHASH 계산에 필요한 값을 미리 계산하는 lea_gcm_init 함수, nonce로부터 초기 카운터를 계산하는 lea_gcm_set_ctr 함수, AAD가 있을 경우 AAD의 GHASH 값을 계산하는 lea_gcm_set_aad 함수, 추가되는 입력 데이터들에 대해서 순차적으로 암호화 또는 복호화를 수행하는 lea_gcm_encrypt 및 lea_gcm_decrypt 함수, 그리고 더 이상 입력 데이터가 없을 때 인증값을 계산하거나 검증하는 lea_gcm_final로 이루어져 있다. GCM 모드 암호화 및 복호화 과정은 다음과 같이 도시할 수 있다.



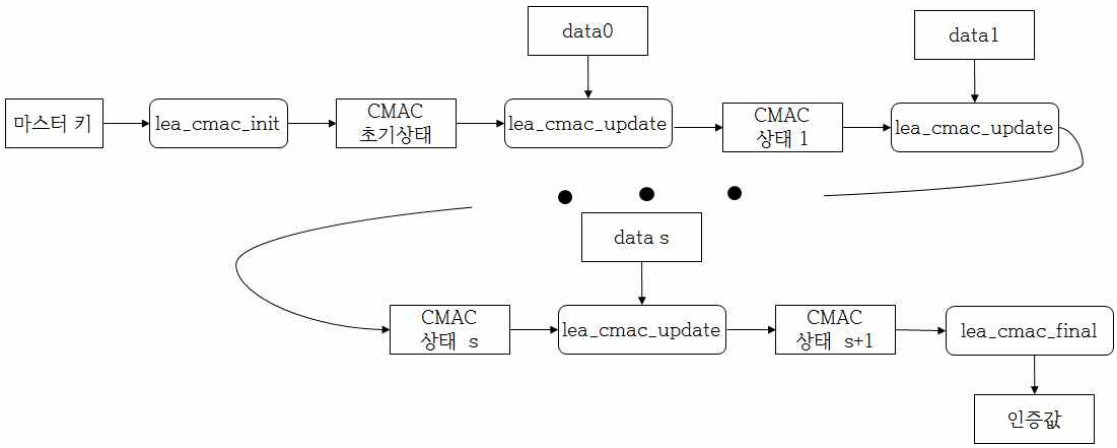
<GCM 모드 암호화 과정>



<GCM 모드 복호화 과정>

CMAC

CMAC도 GCM 모드와 마찬가지로 online 처리를 지원하여 적용 시스템이 제공하는 메모리보다 큰 데이터도 처리할 수 있도록 구현되어 있다. CMAC 인증값 생성과정에서는 라운드 키 생성, subkey 생성 등을 수행하는 초기화 과정 lea_cmac_init 함수를 수행하여 초기상태를 계산하고, 추가되는 입력 데이터들에 대해서 순차적으로 lea_cmac_update를 수행한 후, 더 이상 입력 데이터가 없을 때 lea_cmac_final을 수행하여 인증값을 계산한다. CMAC 동작 과정은 다음과 같이 도시할 수 있다.



<CMAC 인증값 생성 과정>

CMAC에서 인증값 검증 과정은 CMAC 생성 과정을 통해 계산한 인증값과 주어진 인증값이 일치하는지를 확인하는 것이다.

4.3.1. lea_set_key

LEA 라운드 키를 생성한다.

문법

```
void lea_set_key(
        LEA_KEY *key,
        const unsigned char *mk,
        unsigned int mk_len
);
```

매개 변수

key [out]

LEA 라운드 키와 라운드 수 정보를 가진 LEA_KEY 구조체의 주소

mk [in]

마스터키

mk_len [in]

마스터키의 길이(바이트). 입력 값으로 16, 24, 32만 가능하다.

설명

입력된 마스터키를 이용하여 라운드 키를 계산하고 LEA_KEY 구조체를 생성한다. LEA_KEY 구조체는 마스터키 길이 정보와 라운드 키를 포함한다. LEA_KEY 구조체는 사용자가 선언한 후 입력해야 하며 초기화하지 않은 구조체의 주소를 입력해도 정상 동작한다.

4.3.2. lea_ecb_enc

입력된 평문을 LEA를 이용해 ECB 모드로 암호화한다.

문법

```
void lea_ecb_enc(
        unsigned char *ct,
        const unsigned char *pt,
        unsigned int pt_len,
        const LEA_KEY *key
);
```

매개 변수

ct [out]

암호문

pt [in]

암호화하려는 평문

pt_len [in]

평문의 길이(바이트). 16의 배수의 길이만 입력 가능하다.

key [in]

lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 평문을 ECB 모드를 이용해 암호화한다. 평문의 바이트 길이는 16의 배수이어야 한다.

4.3.3. lea_ecb_dec

입력된 암호문을 LEA를 이용해 ECB 모드로 복호화한다.

문법

```
void lea_ecb_dec(
        unsigned char *pt,
        const unsigned char *ct,
        unsigned int ct_len,
        const LEA_KEY *key
);
```

매개 변수

pt [out]

평문

ct [in]

복호화하려는 암호문

ct_len [in]

암호문의 길이(바이트). 16의 배수의 길이만 입력 가능하다.

key [in]

lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 암호문을 ECB 모드를 이용해 복호화한다. 암호문의 바이트 길이는 반드시 16의 배수이어야 한다.

4.3.4. lea_cbc_enc

입력된 평문을 LEA를 이용해 CBC 모드로 암호화한다.

문법

```
void lea_cbc_enc(
        unsigned char *ct,
        const unsigned char *pt,
        unsigned int pt_len,
        const unsigned char *iv,
        const LEA_KEY *key
);
```

매개 변수

ct [out]
: 암호문

pt [in]
: 암호화하려는 평문

pt_len [in]
: 평문의 길이(바이트). 16의 배수의 길이만 입력 가능하다.

iv [in]
: CBC 모드에서 사용할 IV. 16바이트를 입력해야 한다.

key [in]
: lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 평문을 CBC 모드를 이용해 암호화한다. 평문의 바이트 길이는 16의 배수이어야 한다.

4.3.5. lea_cbc_dec

입력된 평문을 LEA를 이용해 CBC 모드로 복호화한다.

문법

```
void lea_cbc_dec(
        unsigned char *pt,
        const unsigned char *ct,
        unsigned int ct_len,
        const unsigned char *iv,
        const LEA_KEY *key
);
```

매개 변수

pt [out]
: 평문

ct [in]
: 복호화하려는 암호문

ct_len [in]
: 암호문의 길이(바이트). 16의 배수의 길이만 입력 가능하다.

iv [in]
: CBC 모드에서 사용할 IV. 16바이트를 입력해야 한다.

key [in]
: lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 암호문을 CBC 모드를 이용해 복호화한다. 복호화 시 암호문의 바이트 길이는 반드시 16의 배수이어야 한다.

4.3.6. lea_ctr_enc

입력된 평문을 LEA를 이용해 CTR 모드로 암호화한다.

문법

```
void lea_ctr_enc(
        unsigned char *ct,
        const unsigned char *pt,
        unsigned int pt_len,
        unsigned char *ctr,
        const LEA_KEY *key
);
```

매개 변수

ct [out]

암호문

pt [in]

암호화하려는 평문

pt_len [in]

평문의 길이(바이트).

ctr [in, out]

CTR 모드에서 사용할 초기 카운터. 16바이트를 입력해야 한다. CTR 모드 연산 후 갱신된다.

key [in]

lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 평문을 CTR 모드를 이용해 암호화한다. CTR 모드 암호화 시 평문의 바이트 길이는 임의의 값을 가질 수 있다.

4.3.7. lea_ctr_dec

입력된 평문을 LEA를 이용해 CTR 모드로 복호화한다.

문법

```
void lea_ctr_dec(
        unsigned char *pt,
        const unsigned char *ct,
        unsigned int ct_len,
        unsigned char *ctr,
        const LEA_KEY *key
);
```

매개 변수

pt [out]

평문

ct [in]

복호화하려는 암호문

ct_len [in]

암호문의 길이(바이트).

iv [in]

CTR 모드에서 사용할 초기 카운터. 16바이트를 입력해야 한다. CTR 모드 연산 후 갱신된다.

key [in]

lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 암호문을 CTR 모드를 이용해 복호화한다. CTR 모드 복호화 시 암호문의 바이트 길이는 임의의 값을 가질 수 있다.

4.3.8. lea_cfb128_enc

입력된 평문을 LEA를 이용해 CFB 모드로 암호화한다.

문법

```
void lea_cfb128_enc(
        unsigned char *ct,
        const unsigned char *pt,
        unsigned int pt_len,
        const unsigned char *iv,
        const LEA_KEY *key
);
```

매개 변수

ct [out]
: 암호문

pt [in]
: 암호화하려는 평문

pt_len [in]
: 평문의 길이(바이트).

iv [in]
: CFB 모드에서 사용할 IV. 16바이트를 입력해야 한다.

key [in]
: lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 평문을 CFB 모드를 이용해 암호화한다. CFB 모드 암호화 시 평문의 바이트 길이는 임의의 값을 가질 수 있다.

4.3.9. lea_cfb128_dec

입력된 평문을 LEA를 이용해 CFB 모드로 복호화한다.

문법

```
void lea_cfb128_dec(
        unsigned char *pt,
        const unsigned char *ct,
        unsigned int ct_len,
        const unsigned char *iv,
        const LEA_KEY *key
);
```

매개 변수

pt [out]
: 평문

ct [in]
: 복호화하려는 암호문

ct_len [in]
: 암호문의 길이(바이트).

iv [in]
: CFB 모드에서 사용할 IV. 16바이트를 입력해야 한다.

key [in]
: lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 암호문을 CFB 모드를 이용해 복호화한다. CFB 모드 복호화 시 암호문의 바이트 길이는 임의의 값을 가질 수 있다.

4.3.10. lea_ofb_enc

입력된 평문을 LEA를 이용해 OFB 모드로 암호화한다.

문법

```
void lea_ofb_enc(
        unsigned char *ct,
        const unsigned char *pt,
        unsigned int pt_len,
        const unsigned char *iv,
        const LEA_KEY *key
);
```

매개 변수

ct [out]
: 암호문

pt [in]
: 암호화하려는 평문

pt_len [in]
: 평문의 길이(바이트).

iv [in, out]
: OFB 모드에서 사용할 IV. 16바이트를 입력해야 한다. 본 함수 수행 후 갱신된다.

key [in]
: lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 평문을 OFB 모드를 이용해 암호화한다. OFB 모드 암호화 시 평문의 바이트 길이는 임의의 값을 가질 수 있다.

4.3.11. lea_ofb_dec

입력된 평문을 LEA를 이용해 OFB 모드로 복호화한다.

문법

```
void lea_ofb_dec(
        unsigned char *pt,
        const unsigned char *ct,
        unsigned int ct_len,
        const unsigned char *iv,
        const LEA_KEY *key
);
```

매개 변수

pt [out]
: 평문

ct [in]
: 복호화하려는 암호문

ct_len [in]
: 암호문의 길이(바이트).

iv [in, out]
: OFB 모드에서 사용할 IV. 16바이트를 입력해야 한다. 본 함수 수행 후 갱신된다.

key [in]
: lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 암호문을 OFB 모드를 이용해 복호화한다. OFB 모드 복호화 시 암호문의 바이트 길이는 임의의 값을 가질 수 있다.

4.3.12. lea_online_init

입력된 키와 운영모드 정보를 이용하여 온라인 구조체를 초기화한다.

문법

```
int lea_online_init(
        LEA_ONLINE_CTX *ctx,
        unsigned int encType,
        const unsigned char *mk,
        int mk_len,
);
```

매개 변수

ctx [out]

온라인 암·복호화에 사용할 LEA_ONLINE_CTX 구조체의 주소

encType[in]

사용할 운영모드.

LEA_ECB_NOPAD_ENC,	LEA_ECB_NOPAD_DEC
LEA_ECB_PKCS5PAD_ENC,	LEA_ECB_PKCS5PAD_DEC
LEA_CBC_NOPAD_ENC,	LEA_CBC_NOPAD_DEC
LEA_CBC_PKCS5PAD_ENC,	LEA_CBC_PKCS5PAD_DEC
LEA_CTR_ENC,	LEA_CTR_DEC
LEA_OFB_ENC,	LEA_OFB_DEC
LEA_CFB128_ENC,	LEA_CFB128_DEC

중에서 선택할 수 있다.

mk [in]

마스터키

mk_len [in]

마스터키의 길이(바이트). 입력 값으로 16, 24, 32만 가능하다.

설명

입력된 마스터키와 운영모드 정보를 이용하여 LEA_ONLINE_CTX 구조체를 초기화한다. 본 함수는 온라인 암·복호화 시 가장 먼저 실행되어야 한다. 에러가 발생하였을 경우 음수를, 정상 동작하였을 경우 0 이상의 정수를 반환한다. ECB 모드와 CBC 모드의 경우에는 PKCS5 패딩을 적용할 수 있으나, 패딩을 사용하지 않는 것으로 설정할 경우 암호화(복호화) 시 전체 암호문의 길이가 16바이트의 배수이어야 정상 동작한다.

4.3.13. lea_online_init_ex

입력된 키와 모드 정보를 이용하여 온라인 구조체를 초기화한다.

문법

```
int lea_online_init_ex(
        LEA_ONLINE_CTX *ctx,
        unsigned int encType,
        const LEA_KEY *key
);
```

매개 변수

ctx [out]

온라인 암·복호화에 사용할 LEA_ONLINE_CTX 구조체의 주소

encType[in]

사용할 모드를

LEA_ECB_NOPAD_ENC,	LEA_ECB_NOPAD_DEC
LEA_ECB_PKCS5PAD_ENC,	LEA_ECB_PKCS5PAD_DEC
LEA_CBC_NOPAD_ENC,	LEA_CBC_NOPAD_DEC
LEA_CBC_PKCS5PAD_ENC,	LEA_CBC_PKCS5PAD_DEC
LEA_CTR_ENC,	LEA_CTR_DEC
LEA_OFB_ENC,	LEA_OFB_DEC
LEA_CFB128_ENC,	LEA_CFB128_DEC

중에서 선택할 수 있다.

key [in]

lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 마스터키와 운영모드 정보를 이용하여 LEA_ONLINE_CTX 구조체를 초기화한다. lea_online_init 과는 달리 lea_set_key 함수를 통해 설정된 LEA_KEY를 사용한다. 본 함수는 온라인 암·복호화시 가장 먼저 실행되어야 한다. 에러가 발생하였을 경우 음수를, 정상 동작하였을 경우 0 이상의 정수를 반환한다. ECB 모드와 CBC 모드의 경우에는 PKCS5 패딩을 적용할 수 있으나, 패딩을 사용하지 않는 것으로 설정할 경우 암호화(복호화) 시 전체 암호문의 길이가 16바이트의 배수이어야 정상 동작한다.

4.3.14. lea_online_update

주어진 데이터를 이용하여 온라인 암·복호화를 수행한다.

문법

```
int lea_online_update(
        LEA_ONLINE_CTX *ctx,
        unsigned char *out,
        const unsigned char *in,
        int in_len
);
```

매개 변수

ctx [in, out]

LEA_ONLINE_CTX 구조체의 주소값

out [out]

암·복호화한 데이터

in [in]

암·복호화할 데이터

in_len [in]

암·복호화할 데이터의 길이(바이트)

설명

주어진 데이터를 이용하여 온라인 암·복호화를 수행한다. 정상적으로 암·복호화가 되었을 경우, out에 기록된 바이트 수를 반환하며, 에러가 발생했을 경우 음수를 반환한다.

lea_online_update 인터페이스는 큰 데이터를 한 번에 암·복호화할 때와 여러번 나누어 입력해 암·복호화를 수행할 때 같은 결과값을 생성하도록 구현되어 있으나, 여러 블록 길이의 데이터를 입력해야만 SIMD 고속화 연산을 수행한다.

4.3.15. lea_online_final

온라인 암·복호화 시 마지막 블록을 처리 한다.

문법

```
int lea_online_final(
        LEA_ONLINE_CTX *ctx,
        unsigned char *out
);
```

매개 변수

ctx [in, out]

LEA_ONLINE_CTX 구조체의 주소값

out [out]

암·복호화 처리한 마지막 블록이 기록될 주소

설명

온라인 암·복호화 시 마지막 블록을 처리한다. 암·복호화 시 패딩을 사용하도록 지정한 경우, 패딩을 추가하거나 제거하는 작업 또한 수행한다. 암·복호화에 성공한 경우 out에 기록된 바이트 수를 반환하며, 에러가 발생한 경우 음수를 반환한다.

4.3.16. lea_ccm_enc

입력된 평문을 LEA를 이용해 CCM 모드로 암호화한다.

문법

```
int lea_ccm_enc(
        unsigned char *ct,
        unsigned char *T,
        const unsigned char *pt,
        unsigned int pt_len,
        unsigned int Tlen,
        const unsigned char *N,
        unsigned int Nlen,
        const unsigned char *A,
        unsigned int Alen,
        const LEA_KEY *key
);
```

매개 변수

ct [out]
: 암호문

T [out]
: 인증값(tag)

pt [in]
: 암호화하려는 평문

pt_len [in]
: 평문의 길이(바이트). 0도 가능하다.

Tlen [in]
: 인증값의 길이(바이트). 4,6,8,10,12,14,16 중의 하나이어야 한다.

N [in]
: nonce 값

Nlen [in]
: nonce 값의 길이(바이트). 7 ~ 13까지 입력 가능하다.

A [in]
: 부가 인증 데이터

Alen [in]

부가 인증 데이터의 길이. 0 입력 시 부가 인증 데이터를 사용하지 않는다.

key [in]

lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 nonce, AAD, 인증값 등을 이용해서 입력된 평문을 CCM 모드로 암호화한다. 암호화에서는 암호문과 인증값을 동시에 생성한다.

4.3.17. lea_ccm_dec

입력된 평문을 LEA를 이용해 CCM 모드로 복호화한다.

문법

```
int lea_ccm_dec(
        unsigned char *pt,
        const unsigned char *ct,
        unsigned int ct_len,
        const unsigned char *T,
        unsigned int Tlen,
        const unsigned char *N,
        unsigned int Nlen,
        const unsigned char *A,
        unsigned int Alen,
        const LEA_KEY *key
);
```

매개 변수

pt [out]
    평문

ct [in]
    암호문

ct_len [in]
    암호문의 길이(바이트). 0도 가능하다.

T [in]
    인증값(tag)

Tlen [in]
    인증값의 길이(바이트). 4,6,8,10,12,14,16 중의 하나이어야 한다.

N [in]
    nonce 값

Nlen [in]
    nonce 값의 길이(바이트). 7 ~ 13까지 입력 가능하다.

A [in]
    부가 인증 데이터

Alen [in]

부가 인증 데이터의 길이. 0 입력 시 부가 인증 데이터를 사용하지 않는다.

key [in]

lea_set_key 함수를 통해 설정된 LEA_KEY 구조체의 주소

설명

입력된 평문을 CCM 모드를 이용해 복호화하고 인증값을 계산하여 주어진 인증값과 일치하는지 비교한다. 일치하지 않을 경우 -1을 반환하고 평문을 모두 0으로 채운다.

4.3.18. lea_gcm_init

입력된 키를 이용하여 LEA_GCM_CTX 구조체를 초기화한다.

문법

```
void lea_gcm_init(
        LEA_GCM_CTX *ctx,
        const unsigned char *mk,
        int mk_len)
```

매개 변수

ctx [out]

GCM 연산에 사용할 LEA_GCM_CTX 구조체의 주소

mk [in]

마스터키

mk_len [in]

마스터키의 길이(바이트). 입력 값으로 16, 24, 32만 가능하다.

설명

입력된 마스터키를 이용하여 LEA_GCM_CTX 구조체를 초기화한다. 라운드 키와 GHASH 계산에 필요한 값을 미리 계산하여 구조체에 저장한다. 본 함수는 GCM 모드 수행 시 가장 먼저 실행되어야 한다.

4.3.19. lea_gcm_set_ctr

IV를 이용하여 GCM 모드에서 사용할 카운터 값을 계산하고 LEA_GCM_CTX 구조체를 갱신한다.

문법

```
void lea_gcm_set_ctr(
        LEA_GCM_CTX *ctx,
        const unsigned char *iv,
        int iv_len)
```

매개 변수

ctx [in]
    갱신 전의 LEA_GCM_CTX 구조체의 주소

iv [in]
    IV

iv_len [in]
    IV의 길이(바이트)

ctx [out]
    갱신 후의 LEA_GCM_CTX 구조체의 주소

설명

입력된 IV를 이용해 GCM 모드에서 사용할 카운터 값을 설정한다. 이 인터페이스는 lea_gcm_init 이후에 실행되어야 하고, lea_gcm_encrypt 또는 lea_gcm_decrypt 인터페이스 호출 전 반드시 선행되어야 한다.

4.3.20. lea_gcm_set_aad

부가 인증 데이터에 GHASH를 수행하고 LEA_GCM_CTX 구조체를 갱신한다.

문법

```
void lea_gcm_set_aad(
        LEA_GCM_CTX *ctx,
        const unsigned char *aad,
        int aad_len)
```

매개 변수

ctx [in]

갱신 전의 LEA_GCM_CTX 구조체의 주소

aad [in]

부가 인증 데이터

aad_len [in]

부가 인증 데이터의 길이(바이트)

ctx [out]

갱신 후의 LEA_GCM_CTX 구조체의 주소

설명

입력된 부가 인증 데이터를 처리한다. 이 인터페이스는 AAD의 길이가 0이 아닐 경우 사용하며, lea_gcm_init 호출 이후에, 그리고 lea_gcm_encrypt 또는 lea_gcm_decrypt 인터페이스 호출 전에 사용하여야 한다.

4.3.21. lea_gcm_encrypt

추가로 입력되는 평문을 GCM 모드로 암호화하고 LEA_GCM_CTX 구조체를 갱신한다.

문법

```
void lea_gcm_encrypt(
        LEA_GCM_CTX *ctx,
        unsigned char *ct,
        const unsigned char *pt,
        int pt_len)
```

매개 변수

ctx [in]

갱신 전의 LEA_GCM_CTX 구조체의 주소

ct [out]

암호문

pt [in]

암호화하려는 평문

pt_len [in]

평문의 길이(바이트). 0도 가능하다.

ctx [out]

갱신 후의 LEA_GCM_CTX 구조체의 주소

설명

입력된 평문을 GCM 모드로 암호화한다. 이 인터페이스를 수행하기 위해서는 lea_gcm_init, lea_gcm_set_ctr 인터페이스가 선행되어야 하고, AAD의 길이가 0이 아닌 경우에는 lea_gcm_set_aad 인터페이스도 선행되어야 한다. 또한 이 인터페이스를 통해 갱신된 LEA_GCM_CTX 구조체는 lea_gcm_final 인터페이스 호출 전까지는 lea_gcm_decrypt의 입력으로 사용될 수 없다. GCM 모드 암호화 시 평문의 길이가 0인 경우에도 lea_gcm_encrypt를 한 번은 호출해야 올바른 인증값을 얻을 수 있다.

4.3.22. lea_gcm_decrypt

추가로 입력되는 암호문을 GCM 모드로 복호화하고 LEA_GCM_CTX 구조체를 갱신한다.

문법

```
void lea_gcm_decrypt(
        LEA_GCM_CTX *ctx,
        unsigned char *pt,
        const unsigned char *ct,
        int ct_len)
```

매개 변수

ctx [in]

갱신 전의 LEA_GCM_CTX 구조체의 주소

pt [out]

평문

ct [in]

복호화하려는 암호문

ct_len [in]

암호문의 길이(바이트). 0도 가능하다.

ctx [out]

갱신 후의 LEA_GCM_CTX 구조체의 주소

설명

입력된 암호문을 GCM 모드로 복호화한다. 이 인터페이스를 수행하기 위해서는 lea_gcm_init, lea_gcm_set_ctr 인터페이스가 선행되어야 하고, AAD의 길이가 0이 아닌 경우에는 lea_gcm_set_aad 인터페이스도 선행되어야 한다. 또한 이 인터페이스를 통해 갱신된 LEA_GCM_CTX 구조체는 lea_gcm_final 인터페이스 호출 전까지는 lea_gcm_encrypt의 입력으로 사용될 수 없다.

4.3.23. lea_gcm_final

인증값을 반환하고 LEA_GCM_CTX 구조체를 초기화한다.

문법

```
int lea_gcm_final(
        LEA_GCM_CTX *ctx,
        unsigned char *tag,
        int tag_len)
```

매개 변수

ctx [in]
LEA_GCM_CTX 구조체의 주소

tag [out]
인증값(tag)

tag_len [in]
인증값의 길이(바이트). 4 이상 16 이하이어야 한다.

설명

암호화 시에는 계산된 인증값을 입력값인 길이만큼 반환하고 복호화 시에는 올바른 인증값이 아닐 경우 -1을 반환한다. 이 인터페이스를 호출할 경우 LEA_GCM_CTX 구조체는 초기화된다. 암호화 및 복호화 시 인증값의 바이트 길이가 4 미만이면 인증값을 계산하지 않고, LEA_GCM_CTX 구조체를 초기화한 후 -1을 반환한다.

4.3.24. lea_cmac_init

LEA_CMAC_CTX를 초기화한다.

문법

```
void lea_cmac_init(
        LEA_CMAC_CTX *ctx,
        const unsigned char *mk,
        int mk_len
);
```

매개 변수

ctx [out]

키 정보, 인증값 등의 정보를 포함하는 구조체의 주소

mk [in]

CMAC 알고리즘에 사용할 마스터키

mk_len [in]

마스터키의 길이(바이트). 입력 값으로 16, 24, 32만 가능하다.

설명

CMAC 인터페이스에서 사용할 LEA_CMAC_CTX 구조체를 초기화한다. 즉, 마스터키로부터 라운드 키와 subkey들을 계산한다.

4.3.25. lea_cmac_update

추가된 데이터를 이용하여 CMAC 상태값을 갱신한다.

문법

```
void lea_cmac_update(
        LEA_CMAC_CTX *ctx,
        const unsigned char *data,
        int data_len
);
```

매개 변수

ctx [in]
    갱신 전 LEA_CMAC_CTX 구조체의 주소값

data [in]
    추가된 데이터

data_len [in]
    추가된 데이터의 길이(바이트).

ctx [out]
    갱신 후 LEA_CMAC_CTX 구조체의 주소값

설명

추가된 데이터를 이용하여 CMAC 상태값을 갱신한다. lea_cmac_update 인터페이스는 큰 데이터를 한 번에 입력해 MAC 값을 계산할 때와 여러번 나누어 입력해 MAC 값을 계산할 때 같은 MAC 값을 생성하도록 구현되어 있다. lea_cmac_update를 이용하여 메모리의 크기 이상의 데이터의 MAC 값을 계산할 수 있다.

4.3.26. lea_cmac_final

CMAC 상태값으로부터 CMAC 값을 계산해서 반환한다.

문법

```
void lea_cmac_final(
        LEA_CMAC_CTX *ctx,
        unsigned char *cmac,
        int cmac_len
);
```

매개 변수

ctx [in]

갱신 전 LEA_CMAC_CTX 구조체의 주소값

cmac [out]

MAC 값

cmac_len [in]

MAC 값의 길이(바이트). 최소 0에서 최대 16까지 입력 가능하다.

설명

CMAC 값을 계산해서 반환한다. lea_cmac_final 인터페이스를 사용한 후에는 데이터를 더 이상 MAC 값을 계산하는데 추가할 수 없다.

4.4. LEA 단독 C 소스코드의 컴파일 방법

LEA의 단독 C 소스코드는 기본적으로 실행 환경과 독립적으로 수행될 수 있도록 구현되어 있다. 그리고 추가적으로 실행 환경의 자원을 활용할 수 있을 경우 그 자원을 활용해 고속화 할 수 있도록 구현되어 있다. LEA 단독 C 소스코드에는 다음의 기술을 활용해 LEA를 고속화 할 수 있는 기법이 적용되어있다.

기술	분류	적용 부분
SSE2	SIMD	ECB, CTR, CCM, GCM 모드의 암·복호화, CBC 모드의 복호화
AVX2		
XOP		
NEON		
PCLMULQDQ		GCM 모드의 인증값 생성

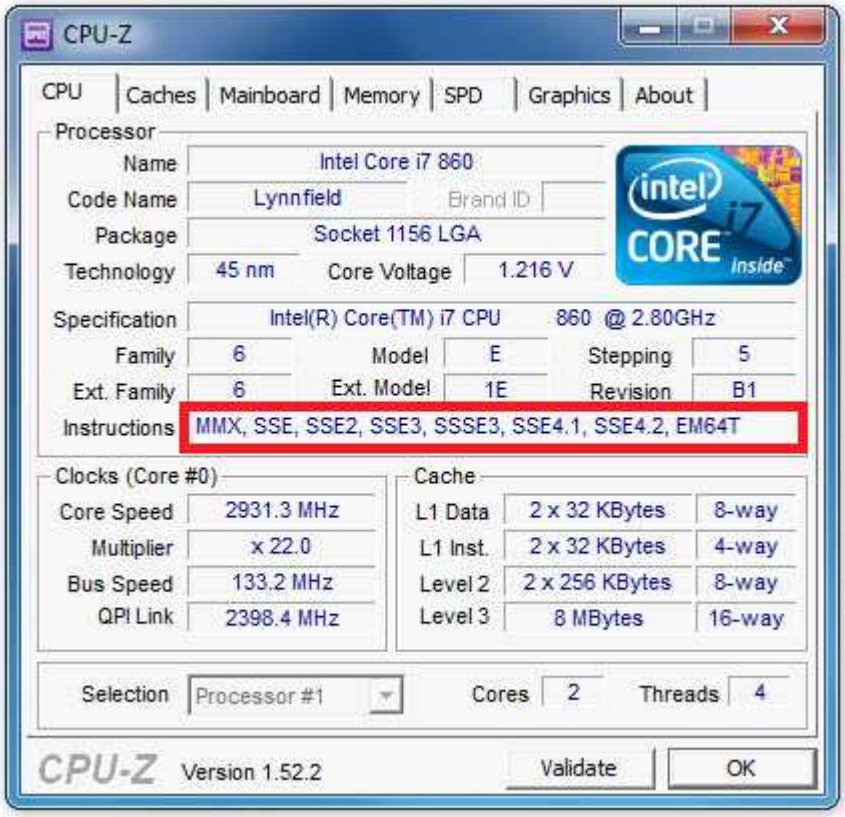
이 기능은 컴파일 시 각 SIMD를 활용한 코드를 탑재하고, 실행 환경에 따라 CPU에서 지원하는 SIMD를 실행하도록 구성되어 있다. 아래에는 대표적인 개발도구인 Visual Studio와 GCC를 이용할 때의 설정방법을 설명한다.

4.4.1. Visual Studio 설정

Visual Studio는 버전에 따라 적용할 수 있는 기술의 종류가 다르다. 또한 이 기술들은 CPU에서 지원해야 적용 가능하다. 다음은 기술별 컴파일 가능한 Visual Studio 버전과 CPU 아키텍처이다. CPU가 지원한다 하더라도 Visual Studio에서 지원하지 않는 기술일 경우, 컴파일 시 해당 SIMD를 사용한 코드가 처리되지 않으므로 주의한다.

기술	Visual Studio 버전	CPU 아키텍처	
		Intel	AMD
SSE2	Visual Studio 6.0 sp5 이상	Pentium 4 이상	Opteron 이상
AVX2	Visual Studio 2013 sp1 이상	Haswell 이상	Excavator 이상
XOP	Visual Studio 2010 sp1 이상	미지원	Bulldozer 이상
PCLMULQDQ	Visual Studio 2008 이상	Westmere 이상	Bulldozer 이상

실행 환경에서 지원하는지는 CPU-Z (http://www.cpuid.com/softwares/cpu-z.html)를 통해서도 확인할 수 있다.



Visual Studio 솔루션 파일 사용

Visual Studio에서 컴파일하기 위해 vs 폴더에 Visual Studio용 솔루션 파일을 제공한다. 솔루션 파일을 이용하면 정적 라이브러리, DLL, 테스트 벡터 프로그램이 생성되며, 생성된 프로그램은 프로그램을 실행하는 환경이 제공하는 SIMD를 선택하여 사용한다.

만약 Visual Studio 2013 SP1 이상을 가지고 있지 않다면, vs2008에 있는 솔루션 파일을 이용하여 컴파일 한다. 이 경우, 해당 Visual Studio에서 제공하는 SIMD만이 지원된다.

정적 및 동적 라이브러리 사용

Visual Studio 2013 SP1 미만의 컴파일러 환경에서는 별도로 제공하는 정적 및 동적 라이브러리 (liblea.lib, dlllea.lib, dlllea.dll)를 사용한다면 AVX2를 적용하는 것이 가능하다. 이 라이브러리들은 Visual Studio 2013 SP1 미만에서도 사용할 수 있다.

별도 프로젝트 사용

제공된 Visual Studio용 솔루션 파일을 이용하지 않는 경우, SSE2, XOP, AVX2를 사용한 파일을 컴파일하기 위해 코드 생성 옵션을 주어야 한다. SIMD를 적용한 C 파일(lea_t_*.c, lea_gcm_pclmul.c)의 속성 -> C/C++ -> 코드 생성으로 들어가 “고급 명령 집합 사용”을 다음과 같이 설정한다.

파일명	기술	고급 명령 집합 사용
lea_t_sse2.c	SSE2	스트리밍 SIMD 확장 2(/arch:SSE2)
lea_core_xop.c	XOP	스트리밍 SIMD 확장 2(/arch:SSE2)
lea_t_xop.c	XOP	스트리밍 SIMD 확장 2(/arch:SSE2)
lea_gcm_pclmul.c	PCLMUL	스트리밍 SIMD 확장 2(/arch:SSE2)
lea_t_avx2_xop.c	AVX2 & XOP	고급 벡터 확장 2 (/arch:AVX2)
lea_t_avx2_sse2.c	AVX2 & SSE2	고급 벡터 확장 2 (/arch:AVX2)

이후 실행코드와 함께 컴파일 시 SSE2, XOP, PCLMUL, AVX2 SIMD 연산을 지원하는 코드로 컴파일 된다.

4.4.2. GCC 설정

GCC도 버전에 따라 적용할 수 있는 기술이 다르다. 다음은 각 기술에 대한 GCC 버전과 필요한 옵션을 정리한 것이다.

기술	GCC 버전	옵션
SSE2	3.0 이상	-msse2
AVX2	4.7 이상	-mavx2
XOP	4.5 이상	-mxop
NEON	4.6 이상	-mfloat-abi=softfp -mfpu=neon
PCLMULQDQ	4.4 이상	-mpclmul -msse4.1

또한 해당 기술은 CPU가 지원해야 적용할 수 있으므로 실행 환경의 CPU가 다음 기술을 지원하는지 확인해야 한다.

기술	CPU 아키텍처		
	Intel	AMD	ARM
SSE2	Pentium 4 이상	Opteron 이상	미지원
AVX2	Haswell 이상	Excavator 이상	미지원
XOP	미지원	Bulldozer 이상	미지원
NEON	미지원	미지원	Cortex-A series
PCLMULQDQ	Westmere 이상	Bulldozer 이상	미지원

제공된 Makefile 사용

GCC에서 컴파일하기 위해 GNU Makefile을 제공한다. GNU Makefile을 이용하면 지정된 cc, ar, as, ranlib을 이용하여 컴파일을 시도한다. GNU Makefile은 POSIX 환경(MinGW, Cygwin, Linux 등)과 x86, x86_64, ARM 환경을 지원하며, 컴파일 시 동적 라이브러리 파일(DLL, SO)과 테스트 벡터를 생성한다. 정적 라이브러리 파일은 “make SHARED=FALSE” 명령을 통해 생성 가능하다. 환경에 따라 GNU make임을 명시하여 make 대신 gmake를 사용하거나, gcc48 등과 같이 CC 변수에 gcc의 버전을 명시해야 할 수도 있다.

사용 예제는 다음과 같다.
- 리눅스 환경에서 동적 라이브러리 파일(so) 생성 및 설치
make install
- 리눅스 환경에서 정적 라이브러리 파일(a) 생성
make SHARED=false
- 리눅스 환경에서 ARM 크로스 컴파일러를 이용한 정적 라이브러리 파일(a) 생성
(사용하는 크로스 컴파일러에 따라 각 변수의 이름이 바뀔 수 있음)
make CC=arm-linux-gnueabi-gcc AS=arm-linux-gnueabi-as AR=arm-linux-gnueabi-ar RANLIB=arm-linux-gnueabi-ranlib
- FreeBSD 환경에서 gcc 4.8 버전을 사용하여 동적 라이브러리 파일(so) 생성 및 설치

gmake CC=gcc48 install
- Solaris 환경에서 OpenCSW로 설치한 gcc 4.8 버전을 사용하여 동적 라이브러리 파일(so) 생성 및 설치
export PATH=/opt/csw/bin:$PATH
gmake install
- SIMD를 사용하지 않는 정적 라이브러리 파일(a) 생성
make TARGET_CPU=generic SHARED=false

정적 라이브러리 만들기

직접 정적 라이브러리를 만들기 위해서는 2단계를 수행하여야 한다. 먼저 소스코드를 원하는 옵션을 적용하여 컴파일하고, 다음으로는 이를 하나의 정적 라이브러리로 통합한다.

1단계를 수행하기 위해 포함시키고자 하는 소스코드와 기능을 선택한 후 각 C 파일을 컴파일한다.
컴파일 명령어는

gcc -c [옵션] [파일 이름]

이다.
-c 옵션은 컴파일은 하되 링킹은 하지 않도록 하는 옵션으로 반드시 포함해야 한다. 이 외에 필요한 옵션이 있다면 [옵션]에 포함시킬 수 있다.
다음은 소스코드를 이용하여 SSE2, PCLMULQDQ만을 지원하는 LEA에 대한 정적 라이브러리를 컴파일하는 예제이다.

gcc -c -O2 lea_core.c lea_base.c cpu_info_ia32.c lea_*_generic.c lea_t_fallback.c -DNO_AVX2 -DNO_XOP
gcc -c -O2 -msse2 lea_t_sse2.c
gcc -c -O2 -msse4.1 -mpclmul lea_gcm_pclmul.c

모든 소스코드를 사용하지 않는 경우 각 파일을 위의 방법으로 컴파일하거나 Makefile을 이용하여 컴파일한다.

2단계로 컴파일한 각 오브젝트를 하나의 정적 라이브러리로 통합한다. 이를 위해 ar 이라는 툴을 사용한다. ar 명령어는

ar r [옵션] [정적 라이브러리 이름] [컴파일한 오브젝트]

이다. r 옵션은 정적 라이브러리가 없을 경우 새로 생성하라는 옵션으로 반드시 포함해야 한다. 그 외에 필요한 옵션을 추가해도 된다.

동적 라이브러리 만들기

직접 동적 라이브러리를 만들기 위해서는 정적 라이브러리를 만들 때와 같이 2단계를 수행하여야 한다.

1단계를 수행하기 위해 포함시키고자 하는 소스코드와 기능을 선택한 후 각 c파일을 컴파일한다.
컴파일 명령어는

gcc -c -fPIC [옵션] [파일 이름]

이다. -c 옵션은 컴파일은 하되 링킹은 하지 않도록 하는 옵션으로 반드시 포함하여야 한다. 또 fPIC 옵션은 Position Independent Code, 즉 위치와 무관하게 동작하는 코드를 생성하여 각 프로그램이 동시에 사용할 수 있도록 해주는 옵션으로 반드시 포함하여야 한다. 다음은 소스코드를 이용하여 AVX2, XOP, SSE2,

PCLMULQDQ를 지원하는 LEA에 대한 동적 라이브러리를 컴파일하는 예제이다.

```
gcc -c -fPIC -O2 lea_core.c lea_base.c cpu_info_ia32.c lea_*_generic.c lea_t_fallback.c
gcc -c -fPIC -O2 -mavx2 -mxop lea_t_avx2_xop.c
gcc -c -fPIC -O2 -mavx2 -msse2 lea_t_avx2_sse2.c
gcc -c -fPIC -O2 -mxop  lea_t_xop.c lea_core_xop.c
gcc -c -fPIC -O2 -msse2  lea_t_sse2.c
gcc -c -fPIC -O2 -msse4.1 -mpclmul lea_gcm_pclmul.c
```

2단계로 컴파일한 각 오브젝트를 하나의 동적 라이브러리로 통합한다. 명령어는

```
gcc -shared -o [동적 라이브러리 이름] [컴파일한 오브젝트]
```

이다.

5. OpenSSL 연동용 LEA C 소스코드

5.1. 서비스

OpenSSL 연동용 LEA C 소스코드는 다음의 서비스를 제공한다.

* High Level Interface(EVP Interface)와 연동
* LEA의 Low Level Interface

각각의 인터페이스에 대한 설명은 http://wiki.openssl.org/index.php/Libcrypto_API 등을 참고할 수 있다.

5.2. 소스코드의 구성

OpenSSL 연동용 LEA C 소스코드는 다음과 같이 구성되어 있다.

파일명	내용
Makefile	컴파일용 Makefile
e_lea.c	EVP Interface, CCM, CMAC 소스코드
lea.h	LEA 인터페이스 명시
lea_locl.h	내부용 헤더
lea_core.c	LEA 소스코드
lea_avx2.c	SIMD 소스코드
lea_sse.c	
lea_neon.c	
lea_xop.c	
lea_ecb.c	운영모드 소스코드
lea_cbc.c	
lea_ctr.c	
lea_cfb.c	
lea_ofb.c	
lea_gcm.c	
lea_misc.c	기타 함수

5.3. 인터페이스

OpenSSL 연동용 LEA 소스코드의 인터페이스는 OpenSSL의 AES 알고리즘 인터페이스와 동일하게 구성되어 있다. 자세한 내용은 http://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption 에서 참고할 수 있다.

5.4. OpenSSL에 연동하는 방법

OpenSSL 연동용 LEA C 소스코드는 LEA를 High Level Interface(EVP Interface)와 Low Level Interface에 간단히 연동할 수 있도록 구성되어 있다. 연동하는 방법은 다음의 4 가지 단계를 거치면 된다.

1. 'lea' 폴더를 다운받은 OpenSSL 경로에서 'openssl/crypto/' 폴더 아래로 옮긴다.
2. openssl/Makefile.org를 열어 'SDIRS' 변수에 lea를 추가한다.

```
# dirs in crypto to build
SDIRS=  \
        objects \
        md2 md4 md5 sha mdc2 hmac ripemd whrlpool \
        des aes rc2 rc4 rc5 idea bf cast camellia seed modes lea \
        bn ec rsa dsa ecdsa dh ecdh dso engine \
        buffer bio stack lhash rand err \
        evp asn1 pem x509 x509v3 conf txt_db pkcs7 pkcs12 comp ocsp ui krb5 \
        cms pqueue ts jpake srp store cmac
```

3. openssl/crypto/objects/objects.txt를 열어 파일 마지막에 다음과 같이 입력한다.

```
# Definition for LEA cipher - ECB, CBC, CTR, OFB, CFB, CCM, GCM, CMAC
!Alias LEA_cipher_suite csor 5
!Alias lea LEA_cipher_suite 1

lea 1                   : LEA-128-ECB           : lea-128-ecb
lea 2                   : LEA-128-CBC           : lea-128-cbc
!Cname lea-128-ofb128
lea 3                   : LEA-128-OFB           : lea-128-ofb
!Cname lea-128-cfb128
lea 4                   : LEA-128-CFB           : lea-128-cfb
lea 6                   : id-lea128-GCM         : lea-128-gcm
lea 7                   : id-lea128-CCM         : lea-128-ccm

lea 21                  : LEA-192-ECB           : lea-192-ecb
lea 22                  : LEA-192-CBC           : lea-192-cbc
!Cname lea-192-ofb128
lea 23                  : LEA-192-OFB           : lea-192-ofb
!Cname lea-192-cfb128
lea 24                  : LEA-192-CFB           : lea-192-cfb
lea 26                  : id-lea192-GCM         : lea-192-gcm
lea 27                  : id-lea192-CCM         : lea-192-ccm

lea 41                  : LEA-256-ECB           : lea-256-ecb
lea 42                  : LEA-256-CBC           : lea-256-cbc
!Cname lea-256-ofb128
lea 43                  : LEA-256-OFB           : lea-256-ofb
!Cname lea-256-cfb128
lea 44                  : LEA-256-CFB           : lea-256-cfb
lea 46                  : id-lea256-GCM         : lea-256-gcm
lea 47                  : id-lea256-CCM         : lea-256-ccm

                        : LEA-128-CFB1          : lea-128-cfb1
                        : LEA-192-CFB1          : lea-192-cfb1
                        : LEA-256-CFB1          : lea-256-cfb1
                        : LEA-128-CFB8          : lea-128-cfb8
                        : LEA-192-CFB8          : lea-192-cfb8
                        : LEA-256-CFB8          : lea-256-cfb8
                        : LEA-128-CTR           : lea-128-ctr
                        : LEA-192-CTR           : lea-192-ctr
                        : LEA-256-CTR           : lea-256-ctr
```

4. OpenSSL 폴더에서 ./config 등을 수행해 컴파일 환경 및 변경사항을 적용하고 make 및 make install을 수행한다. MinGW 등의 경우 make dclean, make depend, make를 사전에 적용 후 컴파일해야 할 수도 있다.

* 타겟 환경이 다른 경우 다음과 같이 입력하여 ./config를 대신할 수 있다.
 - Cygwin 32비트 : ./Configure Cygwin
 - Cygwin 64bit : ./Configure Cygwin64
 - MinGW 32비트 : perl Configure mingw
 - MinGW 64비트 : perl Configure mingw64
 - ARMv7 Linux : (예시)
   export CC=arm-linux-gnueabi-gcc
   export AR=arm-linux-gnueabi-ar
   export AS=arm-linux-gnueabi-as
   export LD=arm-linux-gnueabi-ld
   export RANLIB=arm-linux-gnueabi-ranlib
   export MAKEDEPPROG=arm-linux-gnueabi-gcc

./Configure linux-armv4 --openssldir=/tmp/ossl_arm -march=armv7-a -D__ARM_MAX_ARCH__=7

6. Java 소스코드

6.1. 서비스

LEA의 Java 소스코드는 다음의 서비스를 제공한다.

- LEA의 암호화 운영모드(ECB, CBC, CTR, CFB, OFB)
- LEA의 인증 암호화 운영모드(CCM, GCM)
- LEA 기반의 메시지 인증 코드(CMAC)

6.2. 소스코드의 구성

LEA의 Java 소스코드는 다음과 같이 구성되어 있다.

파일명	내용
BlockCipher.java	블록암호에 사용되는 공통 함수를 정의한 추상 클래스
BlockCipherMode.java	블록암호 운영모드의 공통요소를 모아둔 추상 클래스
BlockCipherModeAE.java	인증 암호화 운영모드의 공통요소를 모아둔 추상 클래스
BlockCipherModeImpl.java	블록암호 운영모드 구현을 위한 추상클래스
BlockCipherModeBlock.java	ECB, CBC 모드 구현을 위한 추상클래스
BlockCihperModeStream.java	CTR, CFB, OFB 모드 구현을 위한 추상 클래스
Mac.java	MAC 계산에 사용되는 함수 인터페이스
Padding.java	패딩에 사용되는 공통 함수를 정의한 추상 클래스
LEA.java	각 운영모드를 쉽게 사용하기 위한 팩토리 클래스
LeaEngine.java	LEA 한 블록 암/복호 구현
ECBMode.java	ECB 모드 구현
CBCMode.java	CBC 모드 구현
CTRMode.java	CTR 모드 구현
CFBMode.java	CFB 모드 구현
OFBMode.java	OFB 모드 구현
CCMMode.java	CCM 모드 구현
GCMMode.java	GCM 모드 구현
CMac.java	CMAC 구현
PKCS5Padding.java	PKCS5 패딩 구현
Hex.java	디버깅을 위한 문자열 변환 함수 모음
Ops.java	XOR 연산 함수 모음
Pack.java	int <-> byte 변환 함수 모음

6.3. 소스코드 사용법

LEA의 Java 소스코드를 사용하는 방법은 공통적으로 객체 생성, 초기화, 암호화(복호화) 수행, 정리의 4단계를 거친다. 단, LEA ECB 및 CBC 모드의 Java 소스코드는 입력 평문 또는 암호문의 길이가 16바이트의 배수이어야 하며, 그렇지 않을 경우 적당한 패딩 알고리즘을 적용한 후 입력하여야 한다. CCM을 제외한 모든 운영모드에서 데이터를 나누어서 online으로 처리하는 기능도 제공하고 있다. 단, GCM의 복호화 시, 인증값 검증 이전에는 복호화된 평문을 버퍼에 저장해 두고 출력은 하지 않는다.

6.3.1. ECB 모드

코드

```
// 객체 생성
BlockCipherMode cipher = new LEA.ECB();

// 암호화
cipher.init(Mode.ENCRYPT, key);
cipher.setPadding(new PKCS5Padding(16));
ct1 = cipher.update(pt1);
ct2 = cipher.doFinal(pt2);

// 복호화
cipher.init(Mode.DECRYPT, key);
cipher.setPadding(new PKCS5Padding(16));
pt1 = cipher.update(ct1);
pt2 = cipher.doFinal(ct2);
```

설명

LEA 클래스의 ECB 클래스를 통해 ECB 모드의 객체를 얻고, key와 함께 ENCRYPT 또는 DECRYPT로 초기화 한 후, 사용한다.

public void init(Mode mode, byte[] key)

동작

암호화 또는 복호화를 수행하기 위해 키를 입력받아 라운드 키를 계산한다.

매개 변수

Mode mode [in]

암호화의 경우 Mode.ENCRYPT, 복호화의 경우 Mode.DECRYPT를 매개변수로 사용함

byte[] key [in]

암·복호화에 사용될 키

반환값

없음

public void setPadding(Padding padding)

동작

암호화 또는 복호화에 사용할 패딩 알고리즘을 설정한다. 기본 설정은 패딩을 사용하지 않는 것이다.

매개 변수

Padding padding [in]

사용할 패딩 알고리즘. null 인 경우는 패딩을 사용하지 않는 것으로 간주하며 Padding 의 생성자로 들어가는 숫자는 블록암호의 한 블록 바이트 길이로, LEA의 경우 16이다. 본 Java 소스코드에서는 PKCS5 패딩을 지원하며, 다른 방식의 패딩은 사용자가 구현하여야 한다.

반환값

없음

public byte[] update(byte[] msg)

동작

입력받은 메시지의 길이가 블록길이의 배수가 아닌 경우 블록길이 배수만큼 처리하고 남은 부분을 버퍼에 남겨두어 다음 메시지가 들어올 때 같이 처리한다.

매개 변수

byte[] msg [in]

암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

계산을 완료한 블록에 대한 암호문 혹은 평문을 반환한다.

public byte[] doFinal(byte[] msg)

동작

버퍼에 남아있는 메시지와 새로 입력받은 메시지를 연접하여 암호화 또는 복호화한다. 이 때 연접한 메시지의 길이가 블록길이(16)의 배수가 아니고 패딩을 하도록 설정되어 있지도 않으면 llegalStateException 이 발생한다.

매개 변수

byte[] msg [in]

암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

마지막 암호문 또는 평문을 반환한다.

public byte[] doFinal()

동작

버퍼에 남아있는 메시지를 암호화 또는 복호화 한다. ECB 모드에서는 입력 메시지의 길이가 블록의 배수이어야 하므로 이 함수를 호출하는 시점에 버퍼에 남아있는 메시지가 있고 패딩을 하도록 설정되어 있지도 않으면 llegalStateException이 발생한다.

매개 변수

없음

반환값

마지막 암호문 또는 평문을 반환한다.

6.3.2. CBC 모드

코드

```
// 객체 생성
BlockCipherMode cipher = new LEA.CBC();

// 암호화
cipher.init(Mode.ENCRYPT, key, iv);
cipher.setPadding(new PKCS5Padding(16));
ct1 = cipher.update(pt1);
ct2 = cipher.doFinal(pt2);

// 복호화
cipher.init(Mode.DECRYPT, key, iv);
cipher.setPadding(new PKCS5Padding(16));
pt1 = cipher.update(ct1);
pt2 = cipher.doFinal(ct2);
```

설명

LEA 클래스의 CBC클래스를 통해 CBC 모드의 객체를 얻고, key, iv와 함께 ENCRYPT 또는 DECRYPT로 초기화 한 후, 사용한다.

public void init(Mode mode, byte[] key, byte[] iv)

동작

암호화 또는 복호화를 수행하기 위해 키를 입력받아 라운드 키를 계산하고 초기화 벡터를 설정한다.

매개 변수

Mode mode [in]

암호화의 경우 Mode.ENCRYPT, 복호화의 경우 Mode.DECRYPT를 매개변수로 사용함

byte[] key [in]

암·복호화에 사용될 키

byte[] iv [in]

초기화 벡터

반환값

없음

public void setPadding(Padding padding)

동작

암호화 또는 복호화에 사용할 패딩 알고리즘을 설정한다. 기본 설정은 패딩을 사용하지 않는 것이다.

매개 변수

Padding padding [in]

사용할 패딩 알고리즘. null 인 경우는 패딩을 사용하지 않는 것으로 간주하며 Padding 의 생성자로 들어가는 숫자는 블록암호의 한 블록 바이트 길이로, LEA의 경우 16이다. 본 Java 소스코드에서는 PKCS5 패딩을 지원하며, 다른 방식의 패딩은 사용자가 구현하여야 한다.

반환값

없음

public byte[] update(byte[] msg)

동작

입력받은 메시지의 길이가 블록길이의 배수가 아닌 경우 블록길이 배수만큼 처리하고 남은 부분을 버퍼에 남겨두어 다음 메시지가 들어올 때 같이 처리한다.

매개 변수

byte[] msg [in]

암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

계산을 완료한 블록에 대한 암호문 또는 평문을 반환한다.

public byte[] doFinal(byte[] msg)

동작

버퍼에 남아있는 메시지 및 새로 입력받은 메시지를 암호화 또는 복호화 한다. 이 때 최종 메시지의 길이가 블록길이(16)의 배수가 아니고, 패딩을 하도록 설정되어 있지도 않으면 llegalStateException이 발생한다.

매개 변수

byte[] msg [in]

암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

마지막 암호문 또는 평문을 반환한다.

public byte[] doFinal()

동작

버퍼에 남아있는 메시지를 암호화 또는 복호화 한다. CBC 모드에서는 입력 메시지의 길이가 블록의 배수여야 하므로 이 함수를 호출하는 시점에 버퍼에 남아있는 메시지가 있고, 패딩을 하도록 설정되어 있지도 않으면 llegalStateException이 발생한다.

매개 변수

없음

반환값

마지막 암호문 또는 평문을 반환한다.

6.3.3. CTR, CFB, OFB 모드

코드

```
// 객체 초기화
BlockCipherMode cipher = new LEA.CTR(); // CTR 모드
BlockCipherMode cipher = new LEA.CFB(); // CFB 모드
BlockCipherMode cipher = new LEA.OFB(); // OFB 모드

// 암호화
cipher.init(Mode.ENCRYPT, key, iv);
ct1 = cipher.update(pt1);
ct2 = cipher.update(pt2);
ct3 = cipher.doFinal(pt3);

// 복호화
cipher.init(Mode.DECRYPT, key, iv);
pt1 = cipher.update(ct1);
pt2 = cipher.update(ct2);
pt3 = cipher.doFinal(ct3);
```

설명

LEA 클래스의 CTR, CFB, OFB 클래스를 통해 각 모드의 객체를 얻고, key, iv와 함께 ENCRYPT 또는 DECRYPT로 초기화 한 후, 사용한다.

public void init(Mode mode, byte[] key, byte[] iv)

동작

암호화 또는 복호화를 수행하기 위해 키를 입력받아 라운드 키를 계산하고, IV 또는 카운터를 설정한다.

매개 변수

Mode mode [in]

암호화의 경우 Mode.ENCRYPT, 복호화의 경우 Mode.DECRYPT를 매개변수로 사용함

byte[] key [in]

암·복호화에 사용될 키

byte[] iv [in]

IV 또는 초기 카운터

반환값

없음

public byte[] update(byte[] msg)

동작

입력받은 메시지의 길이가 블록길이의 배수가 아닌 경우 블록길이 배수만큼 처리하고 남은 부분(16바이트 미만)을 버퍼에 남겨두어 다음 메시지가 들어올 때 같이 처리한다.

매개 변수

byte[] msg [in]

암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

계산을 완료한 블록에 대한 암호문 혹은 평문을 반환한다.

public byte[] doFinal(byte[] msg)

동작

버퍼에 남아있는 메시지와 새로 입력받은 메시지를 연접하여 암호화 또는 복호화 한다.

매개 변수

byte[] msg [in]

암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

마지막 부분의 암호문 혹은 평문을 반환한다.

public byte[] doFinal()

동작

버퍼에 남아있는 메시지를 암호화 또는 복호화 한다.

매개 변수

없음

반환값

마지막 부분의 암호문 혹은 평문을 반환한다.

6.3.4. CCM/GCM 모드

코드

```
// 객체생성
BlockCipherModeAE cipher = new LEA.CCM(); // CCM 모드
BlockCipherModeAE cipher = new LEA.GCM(); // GCM 모드

// 암호화
cipher.init(Mode.ENCRYPT, key, nonce, tag_len);
cipher.updateAAD(aad)
ct1 = cipher.update(pt1);
ct2 = cipher.update(pt2);
ct3 = cipher.doFinal(pt3);

// 복호화
cipher.init(Mode.DECRYPT, key, nonce, tag_len);
cipher.updateAAD(aad);
cipher.update(ct1);
cipher.update(ct2);
pt = cipher.doFinal(ct3);
```

설명

LEA 클래스의 CCM/GCM 클래스를 사용하여 객체를 생성하고, 초기화 파라미터로, key, nonce, tag_len을 제공해야 한다.

public void init(Mode mode, byte[] key, byte[] nonce, int tag_len)

동작

CCM/GCM 모드 수행을 위한 데이터를 입력받아 필요한 변수들을 초기화한다.

매개 변수

Mode mode [in]

암호화의 경우 Mode.ENCRYPT, 복호화의 경우 Mode.DECRYPT를 매개변수로 사용한다.

byte[] key [in]

암·복호화에 사용될 키

byte[] nonce [in]

nonce,
CCM모드의 경우 nonce의 길이는 7~13 바이트이어야 한다.
GCM모드의 경우 nonce의 길이는 1바이트 이상이어야 한다.
적절한 nonce가 입력되지 않으면 NullPointerException 또는 IllegalArgumentException이 발생한다.

int tag_len [in]

인증값의 길이,
CCM 모드의 경우 4, 6, 8, 10, 12, 14, 16만 허용된다.

GCM 모드의 경우 4~16 만 허용된다.
적절한 인증값의 길이가 입력되지 않으면 IllegalArgumentException이 발생한다.

반환값

없음

public byte[] update(byte[] msg)

동작

GCM 모드의 경우 버퍼에 남아 있는 메시지와 입력받은 메시지를 연접하여 블록 단위로 계산을 수행하고, 남는 메시지를 버퍼에 저장한다. CCM 모드의 경우 입력받은 메시지를 버퍼에 추가하기만 한다.

매개 변수

byte[] msg [in]
암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

CCM/GCM 모드는 최종 인증값 계산이 끝나야 결과를 반환할 수 있으므로 항상 null을 반환한다.

public byte[] doFinal(byte[] msg)

동작

버퍼에 남아 있는 메시지 및 새로 입력받은 메시지를 암호화 또는 복호화 한다.

매개 변수

byte[] msg [in]
암호화 모드인 경우 평문, 복호화 모드인 경우 암호문

반환값

암호문을 계산하는 경우 맨 뒤에 인증값을 연접하여 반환하고, 평문을 계산하는 경우 인증값 검증이 성공하면 평문을, 실패하면 null을 반환한다.

public byte[] doFinal()

동작

버퍼에 남아 있는 메시지를 암호화 또는 복호화 한다.

매개 변수

없음

반환값

암호문을 계산하는 경우 맨 뒤에 인증값을 연접하여 반환하고, 평문을 계산하는 경우 인증값 검증이 성공하면 평문을, 검증이 실패하면 null을 반환한다.

6.3.5. CMAC

코드

```
// 객체 생성
Mac cipher = new LEA.CMAC();

// mac 생성
cipher.init(key);
cipher.update(msg1);
cipher.update(msg2);
mac = cipher.doFinal(msg3);
```

설명

초기화할 때 key 값을 넣어주는 init 함수를 사용해야 한다.

public void init(byte[] key)

동작

CMAC 계산을 위한 키를 입력받아 초기화한다.

매개 변수

byte[] key [in]

CMAC 생성에 사용될 키

반환값

없음

public void update(byte[] msg)

동작

버퍼에 남아 있는 메시지와 입력받은 메시지를 연접하여 블록단위로 CMAC 계산을 수행하고, 남는 메시지를 버퍼에 저장한다.

매개 변수

byte[] msg [in]

CMAC을 계산할 메시지

반환값

없음

public byte[] doFinal(byte[] msg)

동작

버퍼에 남아있는 메시지와 새로 입력받은 메시지로부터 최종 CMAC 값을 계산한다.

매개 변수

byte[] msg [in]
CMAC 계산을 위한 메시지

반환값

CMAC 값

public byte[] doFinal()

동작

버퍼에 남아있는 메시지의 CMAC을 계산한다.

매개 변수

없음

반환값

CMAC 값

6.4. Java 소스코드의 컴파일 방법

다음 클래스를 import 해주어야 한다.

```
import kr.re.nsr.crypto.symm.LEA;
import kr.re.nsr.crypto.BlockCipher.Mode;
import kr.re.nsr.crypto.BlockCipherMode;
import kr.re.nsr.crypto.BlockCipherModeAE;
import kr.re.nsr.crypto.Mac;
```

또는 다음과 같이 한 번에 모든 클래스를 import 할 수 있다.

```
import kr.re.nsr.crypto.symm.LEA;
import kr.re.nsr.crypto.*;
```

7. Python 소스코드

7.1. 서비스

LEA의 Python 소스코드는 Python 2.7 이상, 또는 Python 3.2 이상의 환경에서 다음의 서비스를 제공한다.

- LEA의 암호화 운영모드(ECB, CBC, CTR, CFB, OFB)
- LEA의 인증 암호화 운영모드(CCM, GCM)
- LEA 기반의 메시지 인증 코드(CMAC)

7.2. 소스코드의 구성

LEA의 Python 소스코드는 다음과 같이 구성되어 있다.

파일명	내용
LEA/LEA.py	LEA 기본 블록 암호화가 정의된 클래스
LEA/CipherMode.py	모드의 공통 기능이 정의된 상위 클래스
LEA/ECB.py	LEA ECB 모드 구현
LEA/CBC.py	LEA CBC 모드 구현
LEA/CTR.py	LEA CTR 모드 구현
LEA/CFB.py	LEA CFB 모드 구현
LEA/OFB.py	LEA OFB 모드 구현
LEA/CCM.py	LEA CCM 모드 구현
LEA/GCM.py	LEA GCM 모드 구현
LEA/CMAC.py	LEA CMAC 구현

7.3. 소스코드 사용법

LEA의 Python 소스코드를 사용하는 방법은 운영모드 공통적으로 객체 생성 및 초기화, 암호화(복호화) 수행, 정리의 3단계를 거친다. 각 운영모드의 소스코드는 암호화(복호화) 수행 시 평문 또는 암호문을 여러 개로 나누어 입력하더라도 online 처리할 수 있도록 하여 대용량 데이터 암호화(복호화)를 가능하게 한다. 단, LEA ECB 및 CBC 모드의 소스코드는 입력 평문의 길이가 16바이트의 배수이어야 하며, PKCS5 패딩을 지원한다. 그 이외의 패딩 알고리즘은 사용자가 직접 구현하여 적용하여야 한다.

입출력 시 사용하는 평문, 암호문, 인증값, nonce, IV 등의 기본 자료형은 **bytearray**이며, 해당 자료형 대신에 **Buffer Protocol**, **str**(Python 2), **bytes**(Python 3), **memoryview**와 같은 bytes-like-object, 256 미만의 정수로 이루어진 list를 입력값으로 사용 할 수 있다. **str** 타입도 입력값으로 지정할 수 있으나, 언어 인코딩의 문제가 발생할 수 있으므로, 미리 bytes-like-object로 변환하여 사용할 것을 권장한다.

7.3.1. ECB 모드

코드

평문/암호문 1회 입력 시

```
import LEA

#암호화
leaECB = LEA.ECB(LEA.ENCRYPT_MODE, key)
ct = leaECB.update(pt)
ct += leaECB.final()

#복호화
leaECB = LEA.ECB(LEA.DECRYPT_MODE, key)
pt = leaECB.update(ct)
pt += leaECB.final()
```

평문/암호문 2회 입력 시

```
import LEA

#암호화
leaECB = LEA.ECB(LEA.ENCRYPT_MODE, key)
ct = leaECB.update(pt1)
ct += leaECB.update(pt2)
ct += leaECB.final()

#복호화
leaECB = LEA.ECB(LEA.DECRYPT_MODE, key)
pt = leaECB.update(ct1)
pt += leaECB.update(ct2)
leaECB.final()
```

설명

ECB 클래스 생성자에 암·복호화 모드, key를 입력하여 초기화한 후, 사용한다.

LEA.ECB

매개 변수

bool do_enc

암호화, 복호화 여부 설정 변수

bytearray key

LEA.LEA key

LEA.CipherMode key

마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산 없이 재사용한다.

boolean PKCS5Padding

PKCS5 패딩을 사용하여 암·복호화할 것인지를 설정한다. 지정하지 않을 경우, 별도의 패딩을 사용하지 않는다.

반환 값

LEA.ECB(LEA.CipherMode)

update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.ECB.update

매개 변수

bytearray data

입력 바이트열.

반환 값

bytearray

출력 바이트열. 암·복호화 결과물을 반환한다.

LEA.ECB.final

반환 값

bytearray

PKCS5 패딩을 사용하도록 한 경우, 마지막 블록에 패딩을 처리하여 반환한다.
패딩을 사용하지 않는 경우, 데이터 길이가 16의 배수가 아니면 Exception을 발생시킨다.

7.3.2. CBC 모드

코드

평문/암호문 1회 입력 시

```
import LEA

#암호화
leaCBC = LEA.CBC(LEA.ENCRYPT_MODE, key, iv)
ct = leaCBC.update(pt)
ct += leaCBC.final()

#복호화
leaCBC = LEA.CBC(LEA.DECRYPT_MODE, key, iv)
pt = leaCBC.update(ct)
pt += leaCBC.final()
```

평문/암호문 2회 입력 시

```
import LEA

#암호화
leaCBC = LEA.CBC(LEA.ENCRYPT_MODE, key, iv)
ct = leaCBC.update(pt1)
ct += leaCBC.update(pt2)
ct += leaCBC.final()

#복호화
leaCBC = LEA.CBC(LEA.DECRYPT_MODE, key, iv)
pt = leaCBC.update(ct1)
pt += leaCBC.update(ct2)
pt += leaCBC.final()
```

설명

CBC 클래스 생성자에 암·복호화 모드, key, iv를 입력하여 초기화한 후, 사용한다.

LEA.CBC

매개 변수

bool do_enc
암호화, 복호화 여부 설정 변수

bytearray key
LEA.LEA key
LEA.CipherMode key
마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산 없이 재사용한다.

bytearray iv

IV. 길이는 16바이트이어야 한다.

boolean PKCS5Padding

PKCS5 패딩을 사용하여 암·복호화할 것인지를 설정한다. 지정하지 않을 경우, 별도의 패딩을 사용하지 않는다.

반환 값

LEA.CBC(LEA.CipherMode)

update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.CBC.update

매개 변수

bytearray data

입력 바이트열.

반환 값

bytearray

출력 바이트열. 암·복호화 결과물을 반환한다.

LEA.CBC.final

반환 값

bytearray

PKCS5 패딩을 사용하도록 한 경우, 마지막 블록에 패딩을 처리하여 반환한다.
패딩을 사용하지 않는 경우, 데이터 길이가 16의 배수가 아니면 Exception을 발생시킨다.

7.3.3. CTR 모드

코드

평문/암호문 1회 입력 시

```
import LEA

#암호화
leaCTR = LEA.CTR(LEA.ENCRYPT_MODE, key, iv)
ct = leaCTR.update(pt)
ct += leaCTR.final()

#복호화
leaCTR = LEA.CTR(LEA.DECRYPT_MODE, key, iv)
pt = leaCTR.update(ct)
pt += leaCTR.final()
```

평문/암호문 2회 입력 시

```
import LEA

#암호화
leaCTR = LEA.CTR(LEA.ENCRYPT_MODE, key, iv)
ct = leaCTR.update(pt1)
ct += leaCTR.update(pt2)
ct += leaCTR.final()

#복호화
leaCTR = LEA.CTR(LEA.DECRYPT_MODE, key, iv)
pt = leaCTR.update(ct1)
pt += leaCTR.update(ct2)
pt += leaCTR.final()
```

설명

CTR 클래스 생성자에 암·복호화 모드, key, iv(초기 카운터)를 입력하여 초기화한 후, 사용한다.

LEA.CTR

매개 변수

bool do_enc

암·복호화 여부 설정 변수. CTR 모드에서 해당 변수의 값에 따른 동작 차이는 없다.

bytearray key

LEA.LEA key

LEA.CipherMode key

마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산 없이 재사용한다.

bytearray iv

초기 카운터. 길이는 16바이트이어야 한다.

반환 값

LEA.CTR(LEA.CipherMode)

update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.CTR.update

매개 변수

bytearray data

입력 바이트열.

반환 값

bytearray

출력 바이트열. 암·복호화 결과물을 반환한다.

LEA.CTR.final

반환 값

bytearray

전체 데이터의 바이트 길이가 16의 배수가 아닌 경우, 마지막 블록의 암·복호화를 수행 한다.

7.3.4. CFB 모드

코드

평문/암호문 1회 입력 시

```
import LEA

#암호화
leaCFB = LEA.CFB(LEA.ENCRYPT_MODE, key, iv)
ct = leaCFB.update(pt)
ct += leaCFB.final()

#복호화
leaCFB = LEA.CFB(LEA.DECRYPT_MODE, key, iv)
pt = leaCFB.update(ct)
pt += leaCFB.final()
```

평문/암호문 2회 입력 시

```
import LEA

#암호화
leaCFB = LEA.CFB(LEA.ENCRYPT_MODE, key, iv)
ct = leaCFB.update(pt1)
ct += leaCFB.update(pt2)
ct += leaCFB.final()

#복호화
leaCFB = LEA.CFB(LEA.DECRYPT_MODE, key, iv)
pt = leaCFB.update(ct1)
pt += leaCFB.update(ct2)
pt += leaCFB.final()
```

설명

CFB 클래스 생성자에 암·복호화 모드, key, iv를 입력하여 초기화한 후, 사용한다.

LEA.CFB

매개 변수

bool do_enc

암호화, 복호화 여부 설정 변수

bytearray key

LEA.LEA key

LEA.CipherMode key

마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산 없이 재사용한다.

bytearray iv

IV. 길이는 16바이트이어야 한다.

반환 값

LEA.CFB(LEA.CipherMode)

update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.CFB.update

매개 변수

bytearray data

입력 바이트열.

반환 값

bytearray

출력 바이트열. 암·복호화 결과물을 반환한다.

LEA.CFB.final

반환 값

bytearray

전체 데이터의 바이트 길이가 16의 배수가 아닌 경우, 마지막 블록의 암·복호화를 수행한다.

7.3.5. OFB 모드

코드

평문/암호문 1회 입력 시

```
import LEA

#암호화
leaOFB = LEA.OFB(LEA.ENCRYPT_MODE, key, iv)
ct = leaOFB.update(pt)
ct += leaOFB.final()

#복호화
leaOFB = LEA.OFB(LEA.DECRYPT_MODE, key, iv)
pt = leaOFB.update(ct)
pt += leaOFB.final()
```

평문/암호문 2회 입력 시

```
import LEA

#암호화
leaOFB = LEA.OFB(LEA.ENCRYPT_MODE, key, iv)
ct = leaOFB.update(pt1)
ct += leaOFB.update(pt2)
ct += leaOFB.final()

#복호화
leaOFB = LEA.OFB(LEA.DECRYPT_MODE, key, iv)
pt = leaOFB.update(ct1)
pt += leaOFB.update(ct2)
pt += leaOFB.final()
```

설명

OFB 클래스 생성자에 암·복호화 모드, key, iv를 입력하여 초기화한 후, 사용한다.

LEA.OFB

매개 변수

bool do_enc

암·복호화 여부 설정 변수. OFB 모드에서 해당 변수의 값에 따른 동작 차이는 없다.

bytearray key

LEA.LEA key

LEA.CipherMode key

마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산 없이 재사용한다.

bytearray iv

IV. 길이는 16바이트이어야 한다.

반환 값

LEA.OFB(LEA.CipherMode)

update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.OFB.update

매개 변수

bytearray data

입력 바이트열.

반환 값

bytearray

출력 바이트열. 암·복호화 결과물을 반환한다.

LEA.OFB.final

반환 값

bytearray

전체 데이터의 바이트 길이가 16의 배수가 아닌 경우, 마지막 블록의 암·복호화를 수행 한다.

7.3.6. CCM 모드

코드

평문/암호문 1회 입력 시

```
import LEA

#암호화
leaCCM = LEA.CCM(LEA.ENCRYPT_MODE, key, nonce, aad, taglen, ptlen)
ct = leaCCM.update(pt)
ct += leaCCM.final()

#복호화
leaCCM = LEA.CCM(LEA.DECRYPT_MODE, key, nonce, aad, taglen, ptlen)
leaCCM.update(ct)
pt = leaCCM.final()
```

```
import LEA


#암호화
leaCCM = LEA.CCM(LEA.ENCRYPT_MODE, key, nonce, aad, taglen, ptlen)
ct = leaCCM.update(pt1)
ct += leaCCM.update(pt2)
ct += leaCCM.final()

#복호화
leaCCM = LEA.CCM(LEA.DECRYPT_MODE, key, nonce, aad, taglen, ptlen)
#online 복호화 허용 (True일 경우 데이터를 버퍼에 저장. final 수행 시 한꺼번에 처리)
leaCCM.keep_data(False)
pt = leaCCM.update(ct1)
pt += leaCCM.update(ct2)
pt += leaCCM.final()
```

설명

CCM 클래스 생성자에 암·복호화 모드, key, nonce를 입력하여 초기화한 후, 사용한다.

LEA.CCM

매개 변수

bool do_enc

암·복호화 여부 설정 변수.

bytearray key

LEA.LEA key

LEA.CipherMode key

마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산

없이 재사용한다.

bytearray nonce
nonce. 길이는 7~13바이트이어야 한다.

bytearray aad
부가 인증 데이터. 길이는 62280 바이트 이하(0xff00)이어야 한다.

int tag_len
인증값의 바이트 길이. 4,6,8,10,12,14,16 중의 하나이어야 한다.

int data_len
입력되는 전체 메시지의 바이트 길이.

반환 값

LEA.CCM(LEA.CipherMode)
update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.CCM.update

매개 변수

bytearray data
입력 바이트열. 길이가 0이어도 된다.

반환 값

bytearray
암호화 시에는 암호문을 반환한다.
복호화 시에는 평문을 반환하지 않는다.

LEA.CCM.final

반환 값

bytearray
암호화 시에는 마지막 블록의 암호문에 인증값을 덧붙여 반환한다.
복호화 시에는 전체 블록의 평문을 반환한다.

복호화 시 인증값이 일치하지 않을 경우 LEA.CipherMode.TagError가 발생한다.

7.3.7. GCM 모드

코드

평문/암호문 1회 입력 시

```
import LEA

#암호화
leaGCM = LEA.GCM(LEA.ENCRYPT_MODE, key, nonce, aad, taglen)
ct = leaGCM.update(pt)
ct += leaGCM.final()

#복호화
leaGCM = LEA.GCM(LEA.DECRYPT_MODE, key, nonce, aad, taglen)
leaGCM.update(ct)
pt = leaGCM.final()
```

평문/암호문 2회 입력 시

```
import LEA

#암호화
leaGCM = LEA.GCM(LEA.ENCRYPT_MODE, key, nonce, aad, taglen)
ct = leaGCM.update(pt1)
ct += leaGCM.update(pt2)
ct += leaGCM.final()

#복호화
leaGCM = LEA.GCM(LEA.DECRYPT_MODE, key, nonce, aad, taglen)
#online 복호화 허용 (True일 경우 데이터를 버퍼에 저장. final 수행 시 한꺼번에 처리)
leaGCM.keep_data(False)
leaGCM.update(ct1)
leaGCM.update(ct2)
pt = leaGCM.final()
```

설명

GCM 클래스 생성자에 암·복호화 모드, key, nonce를 입력하여 초기화한 후, 사용한다.

LEA.GCM

매개 변수

bool do_enc

암·복호화 여부 설정 변수.

bytearray key

LEA.LEA key

LEA.CipherMode key

마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산

없이 재사용한다.

bytearray nonce
nonce.

bytearray aad
부가 인증 데이터.

int tag_len
인증값의 바이트 길이. 4 이상 16 이하이어야 한다.

반환 값

LEA.GCM(LEA.CipherMode)
update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.GCM.update

매개 변수

bytearray data
입력 바이트열. 길이가 0이어도 된다.

반환 값

암호화 시에는 암호문을 반환한다.
복호화 시에는 평문을 반환하지 않는다.

LEA.GCM.final

반환 값

bytearray
암호화 시에는 마지막 블록의 암호문에 인증값을 덧붙여 반환한다.
복호화 시에는 전체 블록의 평문을 반환한다.

복호화 시 인증값이 일치하지 않을 경우 LEA.CipherMode.TagError가 발생한다.

7.3.8. CMAC

코드

평문/암호문 1회 입력 시

```
import LEA

leaCMAC = LEA.CMAC(key)
leaCMAC.update(pt)
mac = leaCMAC.final()
```

평문/암호문 2회 입력 시

```
import LEA

leaCMAC = LEA.CMAC(key)
leaCMAC.update(pt1)
leaCMAC.update(pt2)
mac = leaCMAC.final()
```

설명

CMAC 클래스 생성자에 key를 입력하여 초기화한 후, 사용한다.

LEA.CMAC

매개 변수

bytearray key

LEA.LEA key

LEA.CipherMode key

마스터 키, 키 길이는 16바이트, 24바이트 또는 32바이트이어야 하며,
길이가 다를 경우 AttributeError가 발생한다.

key로 **LEA.LEA**, **LEA.CipherMode** 오브젝트를 지정할 경우, 해당 오브젝트의 round key를 계산 없이 재사용한다.

반환 값

LEA.CMAC(LEA.CipherMode)

update, final을 메소드로 제공하는 암호 모드 오브젝트를 반환한다.

LEA.CMAC.update

매개 변수

bytearray data

입력 바이트열. 길이가 0이어도 된다.

LEA.CMAC.final

반환 값

bytearray

출력 바이트열. 계산된 CMAC을 반환한다.

코드 관련 문의: leacipher@nsr.re.kr