논문 14-39B-12-02 The Journal of Korean Institute of Communications and Information Sciences '14-12 Vol.39B No.12
http://dx.doi.org/10.7840/kics.2014.39B.12.822

OpenSSL 상에서 LEA 설계 및 구현

박 기 태◆, 한 효 준*, 이 재 훈°

Design and Implementation of Lightweight Encryption Algorithm on OpenSSL

Gi-tae Park◆, Hyo-joon Han*, Jae-hwoon Lee°

요 약

최근 인터넷 환경에서 보안 서비스가 중요시 되면서 SSL/TLS의 사용은 행정기관뿐만 아니라 개인 홈페이지에서도 증가되고 있는 추세이다. 또한 IETF는 사물인터넷 환경에서도 제한된 성능과 자원을 가진 디바이스들이 보안기능을 위해 사용할 수 있도록 DTLS의 적용을 제안하고 있다. 본 논문에서는 LEA알고리즘을 구현하고 이를 OpenSSL 상에 적용 하였다. 그 결과 구현된 알고리즘은 AES 등과 같은 다른 대칭키 알고리즘들과 비교했을 때 연산 속도에 있어 우수한 성능을 보임을 확인 하였다.

Key Words : SSL, TLS, LEA, OpenSSL, Symmetric Encryption, Network Security

ABSTRACT

Recently, A Security service in Internet environments has been more important and the use of SSL & TLS is increasing for the personel homepage as well as administrative institutions. Also, IETF suggests using DTLS, which can provide a security service to constrained devices with lower CPU power and limited memory space under IoT environments. In this paper, we implement LEA(Lightweight Encryption Algorithm) algorithm and apply it to OpenSSL. The implemented algorithm is compared with other symmetric encryption algorithms such as AES etc, and it shows the superior performance in calculation speed.

Ⅰ. 서 론

넷스케이프사가 인터넷을 전자상거래의 플랫폼으로 활용하기 위해 SSL(Secure Socket Layer)을 개발한 이후에 IETF에서는 SSL 버전 3을 바탕으로 TLS(Transport Layer Security)와 UDP환경에서도 동작 할 수 있는 DTLS(Datagram Transport Layer Security) 프로토콜을 제정하였다[1-3]. OpenSSL은 SSL과 TLS, 그리고 DTLS를 구현한 오픈소스이다[4]. OpenSSL은 새로운 암호 알고리즘과 메시지 인증 코드 그리고 암호 키 교환 알고리즘 등이 TLS를 위한 표준으로 정의될 때마다 이를 적용하여 반영하고 있다.

LEA(Lightweight Encryption Algorithm)는 2012년 국가보안기술연구소에서 개발 된 암호 알고리즘으로써 빅 데이터 또는 사물 인터넷 환경에서 적은 메모리와 저효율의 CPU를 가진 초소형 기기에서도 데이터를 고속으로 그리고 안전하게 암호화할 수 있는 알고리즘이다[5,6]. LEA 알고리즘은 128 비트의 블록 암

◆ First Author : Dongguk University Department of Information Communication Engineering , sina2010@naver.com, 학생회원
° Corresponding Author : Dongguk University Department of Information Communication Engineering, jaehwoon@dongguk.edu, 종신회원
* Dongguk University Department of Information Communication Engineering, 학생회원
논문번호 : KICS2014-10-418, Received October 14, 2014; Revised December 9, 2014; Accepted December 9, 2014

호 알고리즘이며 사용 환경에 따라 128, 192, 또는 256 비트의 비밀키를 이용할 수 있도록 설계 되었다.

LEA 알고리즘은 기존의 AES(Advanced Encryption Standard)와 DES(Data Encryption Standard) 암호 알고리즘에서 사용하는 S-Box를 이용하지 않고 비트 수준의 Addition, Rotation, Xor (ARX) 연산만으로 구성되어 동작하기 때문에 대부분의 범용 소프트웨어 플랫폼에서 고속으로 동작 가능하다[7-10]. 실제 LEA의 스펙에 의하면 LEA 알고리즘이 구동되는 CPU의 성능에 따라 차이가 있지만 AES와 비교하여 1.5~3배의 성능 효율을 보이며 코드의 크기도 AES의 25% 정도로 매우 경량 코드이다. 또한 안전성 면에 있어서 LEA는 현존하는 최신 공격 방법에 대해서 안전하다[11].

본 논문에서는 처음으로 OpenSSL 상에 LEA 암호 알고리즘이 동작할 수 있도록 LEA 암호 알고리즘을 구현하고 이것을 통신 모듈에 적용한다. 본 논문의 구성은 다음과 같다. 2장에서는 LEA 암호 알고리즘을, 3장에서는 OpenSSL의 구조에 대해서 설명한다. 그리고 4장에서는 LEA의 구현 및 OpenSSL 상에서 LEA 알고리즘의 적용 내용 및 동작 결과에 대해서 설명하고, 5장에서 결론을 맺는다.

Ⅱ.LEA 암호 알고리즘

LEA알고리즘은 AES와 동일하게 128비트의 블록 암호 알고리즘으로 128, 192, 256비트의 대칭키를 사용할 수 있다[5,7]. LEA 알고리즘은, AES나 DES 암호 알고리즘에서 사용하는 S-Box를 이용 하지 않고, 비트 수준의 Addition, Rotation Xor (ARX) 연산만으로 구성되어 동작하며, 키 스케줄링에 이용되는 상수 값도 4바이트의 상수 값 8개만을 이용하기 때문에 AES와 비교하여 상대적으로 적은 메모리 공간을 암호화 과정에 이용하게 된다.

LEA의 키 스케줄링과 암호화 과정은, 비밀키의 길에 따라 조금씩 차이가 있으며 사용자가 입력한 비밀키와 그림 1의 상수($\delta[0]$~$\delta[7]$)들을 이용하여 키 스케줄링을 통해 라운드 키 (RK_i)를 생성한다. 라운드키

$\delta[0]$ = c3efe9db, $\delta[1]$ = 44626b02,
$\delta[2]$ = 79e27c8a, $\delta[3]$ = 78df30ec,
$\delta[4]$ = 715ea49e, $\delta[5]$ = c785da0a,
$\delta[6]$ = e04ef22a, $\delta[7]$ = e5c40957.

그림 1. LEA에서 라운드 키 생성을 위해 사용되는 상수 값
Fig. 1. Constant Values for generating round keys

는 192비트로 키 스케줄링을 통해 생성된 라운드키는 각 암호화 라운드과정에서 이용되며, 라운드 횟수는 128비트의 경우 24회, 192비트의 경우 28회 256비트의 경우 32회이다. 그림 2는 라운드 키를 이용하여 암호화 하는 절차를 나타낸다.

라운드의 진행 과정은 그림 3과 같은 ARX 연산을 이용한다.

◎ 기호 설명

ROL_i : 32비트 비트열을 I 비트 좌측 순환 이동

ROR_i : 32비트 비트열 I 비트 우측 순환 이동

$X_i[k]$: 32비트 데이터로, 라운드 함수에서 입력 또는 출력 되는 128비트 데이터 중, "i"번째 라운드의 [32*k ~ 32*(k+1)] 비트들을 의미한다.

$Round^{enc}$: 암호화 라운드 과정을 나타낸다.

입력: 128 비트 평문 P, Nr개의 192 비트 라운드키 $RK_0^{enc}, RK_1^{enc}, \cdots, RK_{Nr-1}^{enc}$
출력: 128 비트 암호문 C

```
1: X0 ← P
2: for i = 0 to Nr-1 do
3:     Xi+1 ← Round^enc(Xi, RKi^enc)
4: end for
5: C ← XNr
```

그림 2. 라운드 키를 이용한 암호화 진행 과정
Fig. 2. Encryption procedure using round key

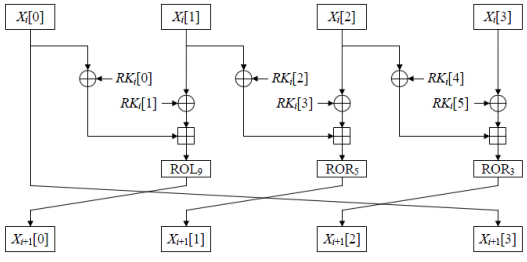


그림 3. 각 암호화 라운드에서의 처리 과정
Fig. 3. Process of LEA encryption round

Ⅲ. OpenSSL 구조

OpenSSL은 Eric A. Young, Tim J. Hudson에 의해 SSL과 TLS, 그리고 DTLS 프로토콜을 C 언어로 구현한 통신 라이브러리로써 윈도우와 리눅스 플랫폼 모두 설치가 가능한 오픈소스이다[4].

OpenSSL에서 지원하고 있는 알고리즘은 표 1과 같다. SSL과 TLS의 보안 통신 과정은 대체로 유사하지만 각 프로토콜에 따라 master secret 계산과 같은

표 1. OpenSSL에서 구현되어 있는 알고리즘
Table 1. Implemented Cryptography Algorithms on OpenSSL

암호 알고리즘	AES, Camellia, DES, IDEA, RC2, RC4, RC5, Triple DES, CAST-128, Blowfish, GOST 28147-89
해시 알고리즘	MD5, MD4, MD2, SHA-1, SHA-2 RIPEMD-160, MDC-2, GOST R 34.11-94
키 교환 알고리즘	RSA, DSA, Diffie-Hellman, Elliptic curve, GOST R 34.10-2001

내부적인 알고리즘의 차이가 있으며, 버전에 따라서 지원하는 키 교환 알고리즘 및 대칭키 암호 알고리즘들의 구성에도 역시 차이가 있다. 이러한 이유로 OpenSSL은 각 프로토콜과 버전들의 동작 차이점을 효율적으로 구현하기 위하여, 크게 통신을 위한 알고리즘과 보안을 위한 암호학적 알고리즘들을 따로 모듈화하여 서로 다른 디렉토리에서 구현하고 있다. 그리고 통신 알고리즘에서도 SSL과 TLS 그리고 DTLS의 공통적인 알고리즘들을 기준으로 하여 각 버전 별로 차이가 있는 알고리즘들은 함수를 따로 작성한 후 해당 함수들을 함수 포인터의 형태로 연결하여 동작하도록 구현하고 있다. 그러므로 LEA 암호 알고리즘을 OpenSSL에 추가하기 위해서는 크게 암호화 라이브러리와 통신 라이브러리에 대한 수정을 필요로 한다. 암호학적 알고리즘들은 [crypto] 디렉토리에 구현되어 있으며 SSL/TLS 핸드쉐이크를 통한 통신 연결 그리고 세션 관리 및 메시지 송수신에 관한 부분은 [ssl] 디렉토리 내에서 구현하고 있다.

3.1 암호화 라이브러리

OpenSSL은 SSL/TLS의 보안 통신의 기능뿐만 아니라 개인키, 공개키, X.509[12] 인증서의 생성, 파일 암/복호화 및 메시지 다이제스트의 계산 기능들을 지원한다. 이러한 기능들을 수행하기 위한 암호학적 알고리즘들은 모두 [crypto] 디렉토리에 구현되어 있다. 또한 보안 통신을 위하여 암호화 모음(Cipher Suite)을 구성해야 할 경우에도 암호화 라이브러리 내에서 구현된 해당 알고리즘 함수들을 호출하여 사용하게 된다. 특히 AES와 DES와 같은 대칭키 알고리즘들의 경우 OpenSSL 내에서 암호화/복호화 기능들을 간편하고 효율적으로 수행하도록 하기 위하여 EVP API라는 라이브러리 형태로 암호화 기능을 제공하고 있다. "EVP"라는 의미는 "Envelop"을 뜻하며 OpenSSL에서 대칭키 암호화 함수들을 총괄하는 인터페이스를 제공한다. 따라서 OpenSSL에서 제공하는 라이브러리 함수 참조를 통한 암호화 프로그램 작성이 필요한 경우 EVP_[암호 알고리즘 명]_[운영모드]()의 형태로 호출하여 사용하거나, 만일 사용하고자 하는 암호 알고리즘의 대칭키 길이가 여러 개인 경우에는 EVP_[암호 알고리즘 명]_[키 길이]_[운영모드]() 형태의 규격화된 이름의 함수들을 호출하여 간단히 구현할 수 있다. 예를 들어 LEA 128 비트 CBC 운영 모드를 사용하여 암호화 프로그램을 작성하고 싶다면 EVP_lea_128_cbc() 함수를 호출하면 된다. 이 함수의 반환 값은 해당 암호 알고리즘의 정보를 포함하고 있는 evp_cipher_st 구조체 변수이며, EVP_EncryptInit()와 EVP_EncryptUpdate(), 그리고 EVP_EncryptFinal() 함수들은 이 변수를 활용하여 라운드 키를 생성하고 입력된 평문을 암호화한다. 그림 4는 EVP API를 이용한 암호화 처리과정을 보여준다.

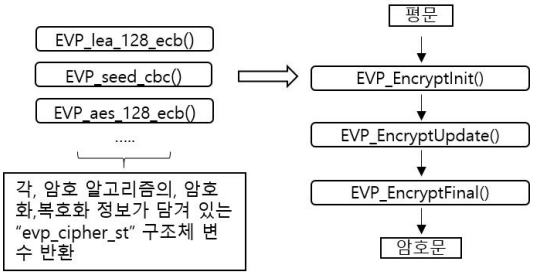


그림 4. EVP API를 이용한 암호화 과정
Fig. 4. Encryption procedure using EVP API

3.2 통신 라이브러리

앞에서 언급한 것과 같이 OpenSSL에서 SSL/TLS 핸드쉐이킹과 같은 통신 알고리즘들은 [ssl] 디렉토리에 구현되어 있다. OpenSSL에서는 SSL과 TLS, 그리고 DTLS의 각 버전들이 모두 구현되어 있기 때문에 공통적인 부분은 "ssl_*"로 시작하는 파일들을 통해 구현하고 있으며 각 버전에 따라 SSLv3는 "s3_," TLS1.0 이후의 버전은 "t1_," DTLS는 "d1_"으로 시작하는 파일명의 파일들에 나누어 구현하고 있다.

그림 5는 OpenSSL을 이용하여 보안 통신 프로그램을 작성하는 과정으로 사용자는 사용하고자 하는 버전의 method() 함수들을 호출하게 된다. 이 함수의 반환 값은 ssl_method_st 구조체로 된 변수이며, 해당 구조체 내에서는 각 버전 별로 다르게 동작하는 알고리즘을 구현한 함수들을 연결한 함수 포인터 변수들로 이루어져 있다. 보안 통신은 method() 함수 호출로 반환된 ssl_method_st 구조체 변수 내의 함수 포인터들과 공통의 라이브러리의 SSL_CTX와 SSL 구조체

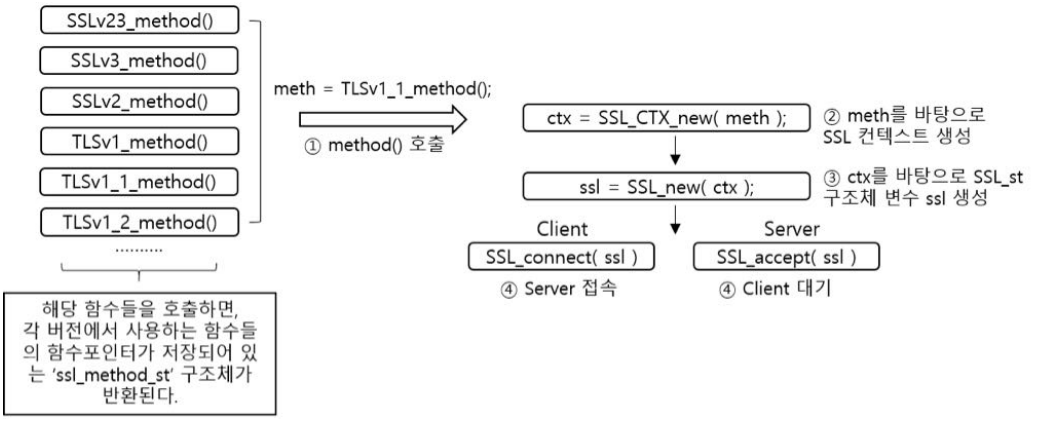


그림 5. 각 버전별 통신 과정
Fig. 5. Communication procedure by each version

및 함수들을 이용하여 구현된다. 그러므로 method() 함수만 변경하면 소스 코드의 큰 수정 없이 모든 SSL/TLS 버전의 보안 통신을 구현할 수 있다.

Ⅳ. LEA 구현 및 OpenSSL에의 적용

본 논문에서는 OpenSSL 1.0.1h 버전을 이용하여 LEA 알고리즘을 구현한다. 앞에서 언급한 것과 같이 OpenSSL은 [crypto] 디렉토리 내에 암호 알고리즘을 구현하기 때문에 LEA 알고리즘을 추가하기 위해서는 [crypto/lea] 디렉토리를 생성하고 "lea" 디렉토리 내에서 LEA 암호화/복호화 알고리즘과 키 스케줄링 함수, 그리고 각 암호 운영 모드 알고리즘들을 구현해야 한다.

표 2는 본 논문에서 [crypto/lea] 디렉토리 내에 구현한 lea 알고리즘 함수들의 원형들이다. OpenSSL 내에서 LEA 알고리즘을 활용하기 위해서는 표 2에서 정의된 함수들을 호출하는 EVP API 함수들을 별도로 추가해 주어야 한다. LEA 알고리즘을 EVP API 형태로 추가하게 되면, OpenSSL 프로그램 내에서 인증서 생성과 보안 통신 그리고 파일 암호화와 같은 기능들에 대해서도 LEA 알고리즘을 활용할 수 있다. 이를 위하여 [crypto/evp] 디렉토리 내에 e_lea.c 파일을 추가하고 EVP_암호명_운영모드() 형태의 함수들을 작성하였다. 한편, OpenSSL에서는 많은 블록 암호 알고리즘들의 중복된 코드를 간략하게하기 위해서 매크로를 활용하는 경우가 많은데. [crypto/evp/evp_locl.h]에 정의된 매크로인 IMPLEMENT_BLICK_CIPHER 매크로를 이용하면 "[암호명]_[운영모드]" 이름으로 evp_cipher_st 구조체 변수를 생성하고, 이 변수 값을 반환하는 EVP_암호명_운영모드() 함수를 간단히 구현할 수 있게 된다.

그림 6은 LEA 알고리즘으로 128 비트의 키를 사용하는 EVP_lea_128_ecb(), EVP_lea_128_cbc(), EVP_lea_128_ofb(), 그리고 EVP_lea_128_cfb() 함수들을 구현하는 매크로를 보여준다.

IMPLEMENT_BLOCK_CIPHER 매크로의 입력으로는 블록 암호 알고리즘의 이름, 평문의 길이, 비밀키의 길이, 키 스케줄링 함수 이름과 NID(Numeric Identifier) 값 등을 인자로 넣어주어야 한다. NID 값은 그림 7에서 볼 수 있듯이 OpenSSL에서 SN(Short Name)과 LN(Long Name)과 함께 각 오브젝트들을 관리하기 위해 할당하는 값으로, [crypto/objects] 디렉토리의 obj_mac.h와 obj_dat.h 파일들에 LEA와 관련된 함수들마다 고유의 NID 값들을 지정해 주어야만 OpenSSL을 올바르게 컴파일 및 설치하여 실행할 수 있다.

```
IMPLEMENT_BLOCK_CIPHER(lea_128, ks, LEA, EVP_LEA_KEY, NID_lea_128,
16, 16, 16, 128, 0, lea_init_key, 0, 0, 0, 0)
```

그림 6. LEA 알고리즘을 EVP API 형식으로 작성하기 위한 매크로
Fig. 6. Macro for transforming LEA to EVP API format

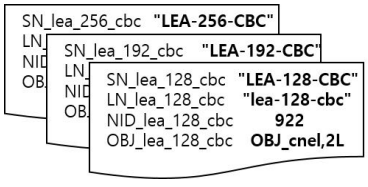


그림 7. obj_mac.h 파일에 추가한 LEA NID값 예시
Fig. 7. Example of obj_mac.h added by NID value

표 2. 본 논문에서 구현한 함수 설명 ('u8'은 unsigned char를 의미)
Table 2. Description of Implemented Functions in this paper

함수	설명
int LEA_set_key(const u8* userKey, const int bits, LEA_KEY* lea_key);	사용자가 입력한, userKey를 바탕으로 하여 lea의 라운드 키 (lea_key)를 생성한다. bits = 키 길이
void LEA_encrypt(const u8* in, u8* out, const LEA_KEY* lea_key);	사용자가 입력한 평문 (in)을 lea_key의 라운드 키를 바탕으로 하여 암호문 (out)으로 변환시킨다.
void LEA_decrypt(const u8 *in, u8* out, const LEA_KEY* lea_key);	사용자가 입력한 암호문 (in)을 lea_key의 라운드 키를 바탕으로 하여 평문 (out)으로 변환시킨다.
void LEA_ecb_encrypt(const u8* in, u8* out, const LEA_KEY *lea_key, const int enc);	enc = 1 : 암호화 , enc = 0 복호화 왼쪽 4개의 함수들은 ecb, cbc, cfb, ofb 모드로 LEA_encrypt() 함수를 이용하여 평문을 암호문으로 암호화 시키거나, LEA_decrypt() 함수를 이용하여 암호문을 평문으로 복호화 시킨다.
void LEA_cbc_encrypt(const u8* in, u8* out, size_t len, const LEA_KEY* lea_key, u8* ivec, int enc);	
void LEA_cfb128_encrypt(const u8* in, u8* out, size_t len, const LEA_KEY* lea_key, u8* ivec, int *num, int enc);	
void LEA_ofb128_encrypt(const u8* in, u8* out, size_t len, const LEA_KEY* lea_key, u8* ivec, int* num);	

4.1 키 스케줄링

LEA 알고리즘은 비밀키의 종류가 128, 192 그리고 256 비트 방식이 존재하기 때문에 각각의 비밀키에 따른 키 스케줄링 과정을 독립적으로 수행하도록 구현해야 한다. 그림 8은 본 논문에서 구현한 LEA 알고리즘의 키 스케줄링 과정을 나타낸다.

evp_cipher_st 구조체는 해당 알고리즘이 사용하는 암호화 함수와 복호화 함수 그리고 키 스케줄링 함수들의 함수 포인터들과 평문의 길이 및 비밀키 길이 등의 정보를 저장하고 있으며, EVP_암호명_운영모드() 호출을 통해 해당 구조체 변수가 반환된다. 본 논문에서는 EVP_lea_[키길이]_[운영모드]()에서 반환 받은 evp_cipher_st 구조체 변수 내에 있는 비밀키 길이 정보를 바탕으로 LEA_set_key()함수에서 라운드 키들을 생성하도록 하였다. LEA_set_key() 함수에서는 lea_key_st 구조체 변수 내에 해당 암호화, 복호화 과정의 라운드 횟수와, 각 라운드 키들이 생성되어 저장된다.

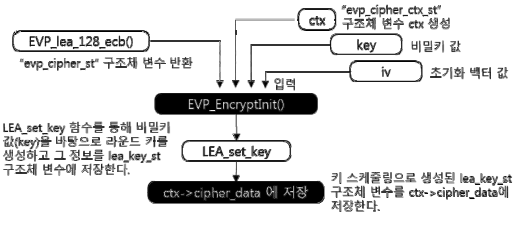


그림 8. EVP API함수를 통한 키 스케줄링 과정
Fig. 8. Key scheduling process using EVP API

4.2 암호 운영 모드

각 암호 운영 모드는 아래 그림 9와 같이 [crypto/modes] 디렉토리의 파일들에서 정의하고 있는 CRYPTO_cbc128_encrypt(), CRYPTO_cfb128_encrypt(), CRYPTO_ofb128_encrypt() 함수들을 이용하여, 본 논문에서 구현한 암호화 함수인 LEA_encrypt()와 복호화 함수인 LEA_decrypt()를 함수 포인터로 연결하여 구현한다.

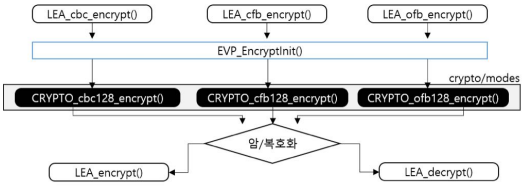


그림 9. 암호 운영모드 구현 방법
Fig. 9. Way of Cipher operation mode implementation

4.3 통신 라이브러리

LEA를 이용한 보안 통신을 구현하기 위해서 암호화 모음(Cipher Suite) 리스트에 LEA암호 알고리즘을 추가해주어야 한다. 본 논문에서는 다른 대칭키 알고리즘처럼 LEA 알고리즘의 비밀키로 128비트와 256비트를 사용 할 수 있도록 Cipher Suite를 구성하였다.

본 논문에서 Cipher Suite 구성을 위해 할당한 값은 그림 10과 같다.

SSL/TLS에서 Cipher Suite를 구성하기 위해서 2바이트를 할당하지만, OpenSSL에서는 버전 정보를 상위 2바이트에 함께 표시하여 총 4바이트로 값을 할당한다. 실제 그림 10의 4바이트 값들 중 하위 2바이트가 Cipher Suite 리스트 구성 시 사용되어지는 값이다.

그림 10에 할당한 값을 이용하여 ClientHello 메시지와 Server Hello 메시지의 Cipher Suite 리스트에 LEA를 포함하여 전송하기 위해서는, [ssl] 디렉토리의 s3_lib.c 파일 내에 정의된 ssl3_ciphers 배열에 LEA 알고리즘을 ssl_cipher_st 구조체의 형태로 추가해주어야 한다.

그림 11은 ssl3_ciphers 배열에 LEA 알고리즘을 이용하여 Cipher Suite를 구성하는 방법을 나타낸 것

128비트 비밀키를 이용한 CipherSuite

TLS1_CK_RSA_WITH_LEA_128_CBC_SHA	0x030000A8
TLS1_CK_DH_DSS_WITH_LEA_128_CBC_SHA	0x030000A9
TLS1_CK_DH_RSA_WITH_LEA_128_CBC_SHA	0x030000AA
TLS1_CK_DHE_DSS_WITH_LEA_128_CBC_SHA	0x030000AB
TLS1_CK_DHE_RSA_WITH_LEA_128_CBC_SHA	0x030000AC
TLS1_CK_ADH_WITH_LEA_128_CBC_SHA	0x030000AD

256비트 비밀키를 이용한 CipherSuite

TLS1_CK_RSA_WITH_LEA_256_CBC_SHA	0x030000AE
TLS1_CK_DH_DSS_WITH_LEA_256_CBC_SHA	0x030000AF
TLS1_CK_DH_RSA_WITH_LEA_256_CBC_SHA	0x030000B0
TLS1_CK_DHE_DSS_WITH_LEA_256_CBC_SHA	0x030000B1
TLS1_CK_DHE_RSA_WITH_LEA_256_CBC_SHA	0x030000B2
TLS1_CK_ADH_WITH_LEA_256_CBC_SHA	0x030000B3

그림 10. LEA를 이용하기 위한 Cipher Suite 할당 값
Fig. 10. Assigned Cipher Suite value for LEA

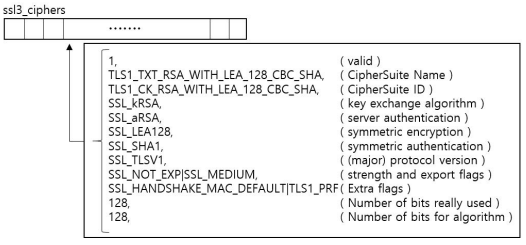


그림 11. ssl_cipher 구조체의 형태로 구성된 SSL3_Ciphers 배열예시
Fig. 11. Example of configuring SSL3_Ciphers array as ssl_cipher_st structures

이다. OpenSSL내에서는 여러 개의 Cipher Suite 중에서 해당 Cipher Suite을 빠른 시간 내에 찾기 위해서 이진탐색을 이용한다. 이진탐색의 기준 값은 Cipher Suite에 할당한 ID 값으로 본 논문에서도 LEA의 Cipher Suite들이 ssl3_ciphers 배열 내에서 ID 값을 기준으로 오름차순으로 정렬되도록 구성하였다.

4.4 암호화 결과 비교

그림 12는 구현한 LEA알고리즘이 OpenSSL의 암호 알고리즘 리스트에 정상적으로 추가되었음을 보여준다. OpenSSL상에서 적용된 구현이 올바른지 확인하기 위하여, LEA의 참조코드[13]와 본 논문에서 구현한 LEA알고리즘의 암호화 복호화 결과를 비교하였다. OpenSSL의 EVP API방식을 이용하였으며, 비밀키는 [13]의 참조코드에서 사용한 비밀키를 동일하게 사용하였다. 각각 ECB 모드로 암호화, 복호화를 수행한 결과, 참조코드와 본 논문에서 구현한 LEA의 암호화 결과가 동일함을 확인 하였다.

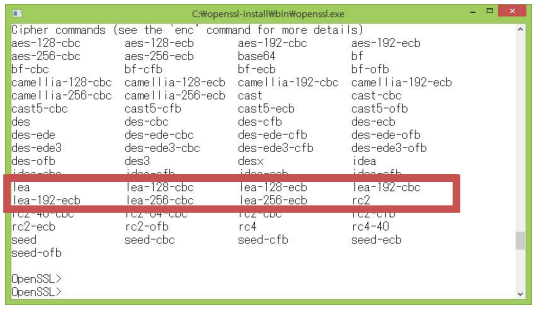


그림 12. OpenSSL에 LEA 알고리즘이 추가된 결과 화면
Fig. 12. Screen of result OpenSSL added LEA algorithm

4.5 Speed 명령어를 이용한 성능 평가

OpenSSL 프로그램 내의 speed 명령어를 이용하여, AES와 DES, SEED[14], CAMELLIA[15], IDEA[16] 그리고 LEA의 알고리즘 처리 속도를 각각 비교해 보았다. speed 명령어는 OpenSSL에 포함된 대칭키 암호 알고리즘의 성능을 비교하기 위한 명령어로, 해당 명령어를 이용하면 16/64/256/1024/8192 바이트 단위의 블록 데이터를 바탕으로 사용자가 선택한 암호 알고리즘 이용하여 3초 동안 암호화하고, 총 암호화한 데이터 양을 비교한다. 리눅스와 윈도우 운영체제에서 모두 실행해 보았으며 결과는 그림 13과 같다.

그림 13에서 볼 수 있듯이 LEA의 암호화/복호화 처리 속도가 다른 알고리즘에 비해 월등히 빠른 것을 확인 할 수 있다. 특히 openssl speed 명령어 기준으로 LEA 알고리즘이 많이 사용되어지는 알고리즘인 AES

(단위 k)

Type	16bytes	64bytes	256bytes	1024bytes	8192bytes
des cbc	77,878.16	78,375.21	77,667.24	77,035.52	76,838.78
des ede3	27,710.49	27,764.80	27,748.95	27,795.80	27,866.45
idea cbc	73,996.41	76,482.17	77,880.49	78,147.58	78,030.84
seed cbc	59,854.05	60,722.15	60,934.06	60,677.95	61,005.82
aes-128 cbc	113,316.70	124,954.85	127,739.31	129,538.39	129,869.06
aes-192 cbc	96,007.15	105,273.17	107,039.98	108,622.85	108,904.45
aes-256 cbc	84,246.09	90,697.34	92,205.14	92,566.88	93,093.89
lea-128 cbc	279,816.26	303,482.24	309,809.32	313,075.37	314,126.46
lea-192 cbc	249,927.86	270,845.18	273,060.40	277,012.48	278,216.70
lea-256 cbc	224,404.75	224,241.49	225,646.08	223,959.35	225,258.15
camellia-128 cbc	118,715.33	150,490.17	161,874.60	164,830.89	165,274.28
camellia-192 cbc	94,227.52	116,155.61	122,253.52	124,546.73	125,119.15
camellia-256 cbc	95,093.69	116,209.69	122,431.32	124,048.58	125,272.06

그림 13. 리눅스 환경에서 비교한 결과 (암호화 양 비교)
Fig. 13. Result of comparison of each algorithms on Linux (comparison of encryption amount)

보다도 처리 속도에 있어 2.4~2.6배정도의 우수함을 보임을 확인 할 수 있다.

4.6 LEA를 이용한 보안 통신 적용 결과

4.4절과 4.5 절의 결과에서, 본 논문의 구현에 문제가 없음을 확인하였으므로 OpenSSL에서 s_server와 s_client 명령어를 이용하여 LEA를 이용한 보안통신을 확인해 보았다. 서버와 클라이언트는 리눅스와 윈도우 각기 다른 환경에서 실행 하였다. 서버의 경우 "-cipher" 옵션을 이용하여, Server Hello의 cipher suite가 항상 "LEA256-SHA"가 선택 되도록 하였다.

그림 14는 WireShark[17]를 이용하여 Client Hello 메시지 패킷을 캡쳐한 결과로써, cipher suite리스트에 LEA256-SHA를 의미하는 "0x00ae"가 포함되었음을 확인 할 수 있다.

그림 15는 Server Hello 메시지를 캡쳐한 것으로, cipher suite로 LEA256-SHA의 "0x00ae"가 포함되었음을 확인 할 수 있고, 이는 서버가 정상적으로 LEA 알고리즘이 구현되어 있음을 의미한다.

그림 14과 15에서 Cipher Suite의 byte값이 "0x00ae"로 되어 있으나, WireShark의 화면 출력에서는 "TLS_PSK_WITH_AES_128_CBC_SHA256"으로 되어 있는데, 실제 RFC4279[18] 상에서의 해당 값은 "0x00BD"이다. 따라서 WireShark에서는 잘못 표현하고 있지만, Client Hello와 Server Hello 모두 "0x00ae"가 포함된 것으로 보아 통신에서도 문제없이 LEA 알고리즘이 적용되었음을 확인 할 수 있다.

```
⊟ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 326
  ⊟ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 322
      Version: TLS 1.2 (0x0303)
    ⊞ Random
      Session ID Length: 0
      Cipher Suites Length: 172
    ⊟ Cipher Suites (86 suites)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
        Cipher Suite: TLS_PSK_WITH_AES_128_CBC_SHA256 (0x00ae)
        Cipher Suite: TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0084)
        Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
        Cipher Suite: TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA (0xc01c)
```

그림 14. client_hello 패킷을 캡쳐한 화면
Fig. 14. Captured screen of client_hello packet

```
⊟ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 58
  ⊟ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 54
      Version: TLS 1.2 (0x0303)
    ⊞ Random
      Session ID Length: 0
      Cipher Suite: TLS_PSK_WITH_AES_128_CBC_SHA256 (0x00ae)
      Compression Method: null (0)
      Extensions Length: 14
    ⊞ Extension: renegotiation_info
```

그림 15. ServerHello 패킷을 캡쳐한 화면
Fig. 15. Captured screen of ServerHello packet

```
New, TLSv1/SSLv3, Cipher is LEA256-SHA
Server public key is 1024 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : LEA256-SHA
    Session-ID: F272CC4B3CD5A4B532DBE85BC1B40A6EDA5859E8E0EBA5239855B88E5348AAE7

    Session-ID-ctx:
    Master-Key: 229D60EA3D1F3071B0D83660C87A304D4F6D4892DBA87B28886E3D56900E6914
19EA6547F1517FFE0EB5281AD59C5CB7
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
```

그림 16. Client의 OpenSSL 콘솔상의 결과 화면
Fig. 16. Result screen of Client's OpenSSL console

그림 16은 실제 Client의 OpenSSL 콘솔 화면으로 LEA256-SHA가 올바르게 Cipher Suite로 선택 되어 Handshake 과정을 마쳤음을 보여준다.

V. 결 론

이 논문에서 우리는 LEA 알고리즘을 구현하고 또한 구현된 알고리즘을 OpenSSL에 적용하였다. LEA 알고리즘이 올바르게 동작한다는 것을 확인하였으며 또한 구현된 LEA 알고리즘이 OpenSSL 상에서 동작하는 것을 확인하였다. 구현된 LEA 알고리즘은 TLS 통신뿐만 아니라 DTLS 통신에서도 올바르게 동작하는 것을 역시 확인하였다. LEA 알고리즘은 AES 암호 알고리즘에 비하여 최소 2.4배 이상의 빠른 속도로 암호화가 이루어져서 AES보다 우수한 성능을 가지고 있으며 또한 낮은 CPU 성능과 적은 메모리 환경에서도 동작할 수 있기 때문에 센서와 같은 제한된 자원을 가진 장치에서도 동작할 수 있다[5]. 향후에는 LEA 알고리즘을 좀 더 빨리 동작할 수 있도록 하기 위한 병렬화 및 하드웨어화에 대한 연구를 계속할 예정이다.

References

[1] A. Freier, P. Karlton, and P. Kocher, *The Secure Sockets Layer (SSL) Protocol Version 3.0*, RFC 6101, Aug. 2011.

[2] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5746, Aug. 2008.

[3] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, RFC 6347, Aug. 2012.

[4] *OpenSSL* from http://openssl.org 2014

[5] D. Hong, J. Lee, D. Kim, D. Kwon, K. H. Ryu, and D. Lee, "LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors" in *Proc. WISA 2013,* pp. 3-27, Jeju Island, Korea, Mar 2014.

[6] J. Park et al., "128-Bit Block Cipher LEA," TTAK.KO-12.0223, Dec. 2013.

[7] NIST, "*Advanced Encryption Standard(AES)*" Federal Information Processing Standard, FIPS PUB 197, Nov. 2001.

[8] National Bureau of Standards "*Data Encryption Standard,*" FIPS PUB 46, Jan. 1987.

[9] B. A. Forouzan, *Cryptography & Network Security,* 1st Ed., Mcgraw-Hill, 2008.

[10] William Stalling, *Cryptography and Network Security,* 5th Ed., Prentice Hall, 2011.

[11] B. Andrey, M. Nicky, T. Elamr, T. Denis, and V. Kerem "Security Evaluation of the Block Cipher LEA Final Report," *COSIC,* Jul. 2011.

[12] D. Cooper et al., *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 5280, May 2008.

[13] Korean Cryptography Forum, *Block Encryption LEA Reference Source Code(2014)*, Retrieved Jun., 2. 2014 from "http://www.kcryptoforum.or.kr/"

[14] J. Park, S. Lee, J. Kim, and J. Lee, *The SEED Encryption Algorithm*, RFC 4009, Feb. 2005.

[15] M. Matsui, J. Nakajima, and S. Moriai, *A Description of the Camellia Encryption Algorithm*, RFC 3713, Apr. 2004.

[16] Schneier and Bruce, "THE IDEA ENCRYPTION ALGORITHM - The International Data Encryption Algorithm (IDEA) may be one of the most secure block algorithms available to the public today. Bruce examines its 128-bit-long key," *Dr. Dobb's journal - Software Tools for the Professional Programmer*, vol. 18, no. 13, pp. 50-57, Oct. 1993.

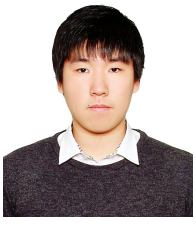
[17] *WireShark* from https://www.wireshark.org/ 2014.

[18] P. Eronen and H. Tschofenig *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*, RFC 4279, Dec. 2005.

박 기 태 (Gi-tae Park)

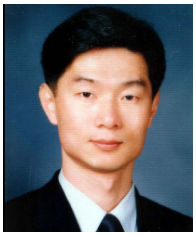
2013년 8월 : 동국대학교 정보통신공학과 졸업(공학사)
2013년 9월~현재 : 동국대학교 정보통신공학과 석사 과정
<관심분야> 정보보호, 네트워크 보안, 사물인터넷

한 효 준 (Hyo-joon Han)



2012년 3월~현재 : 동국대학교 정보통신공학과 학사 과정
<관심분야> 정보보호, 네트워크 보안, 모바일 보안

이 재 훈 (jae-hwoon Lee)



1985년 2월 : 한양대학교 전자공학과 학사
1987년 2월 : 한국과학기술원 전기및전자공학과 석사
1995년 8월 : 한국과학기술원 전기 및 전자공학과 박사
1987년 3월~1990년 4월 : 데이콤 연구원
1990년 9월~1999년 2월 : 삼성전자 정보통신부문 선임연구원
1999년 3월~현재 : 동국대학교 정보통신공학과교수
<관심분야> IP 이동성, 다중 엑세스 프로토콜, 인터넷 프로토콜, 초고속 통신, 네트워크 보안