

➤ 왜 대칭키를 사용?

슬라이드 쇼에서 설명한 바와 같이 빠른 추후에 있을 빠른 데이터 전송에도 적용할 수 있게 하기 위해서이다. 또한 매번 공개키 보다 제공하기 편하기 때문이다.

➤ AES?

대칭키 방식에서 가장 보편적으로 쓰이는 암호화 방식이다. 128, 192, 256비트의 키를 적용할 수 있어 보안성이 뛰어나고 공개된 알고리즘이라 누구나 사용할 수 있다. DES라는 알고리즘의 취약점을 보완하면서 나온 것이다.

고정된 크기의 블록 단위로 암호화 연산을 수행하는 블록 암호 알고리즘이며 라운드 수(암호화 알고리즘(XOR, 덧셈 곱셈 등등)을 수행하는 횟수)는 키 길이에 따라 각각 10, 12, 14이다.

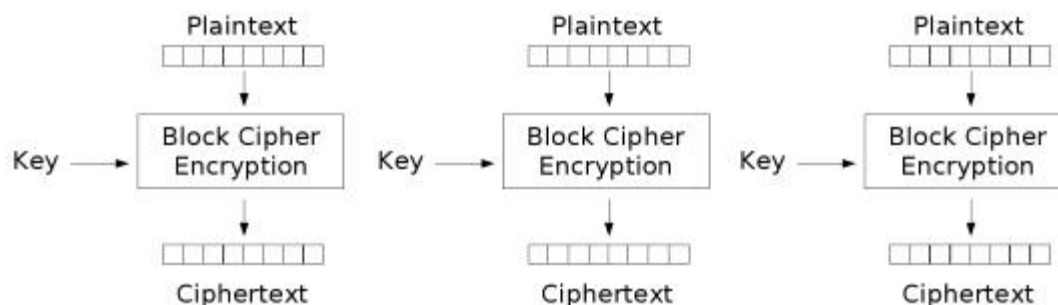
	Key length (Nk words)	Block length (Nb words)	Number of Rounds
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

< Key-Block-Round Combinations >

AES에서 블록으로 암호화를 할 때 4가지 모드가 있다.

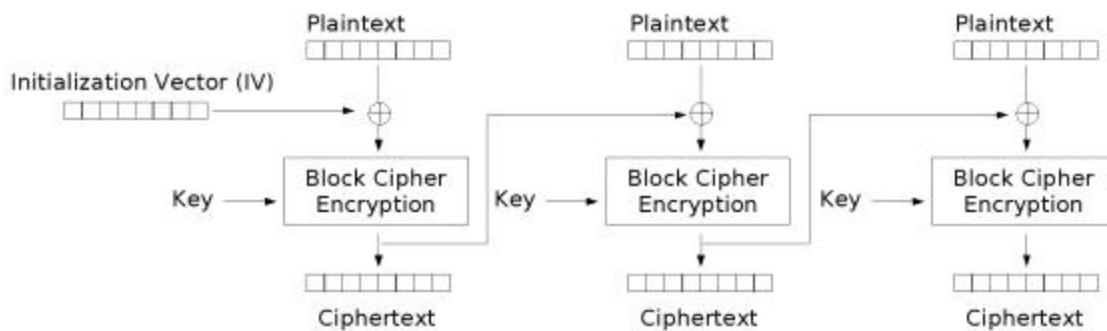
1. ECB (Electric Code Book)
2. CBC (Cipher Block Chaining)
3. OFB (Output Feed Back)
4. CFB (Cipher Feed Back)

첫 번째로 ECB모드는 단순히 한 블록씩 처리를 한다. 암호문 공격에 취약하며 사이즈가 큰 문서의 암호는 어울리지 않아 크게 쓰이고 있지는 않은 듯하다.



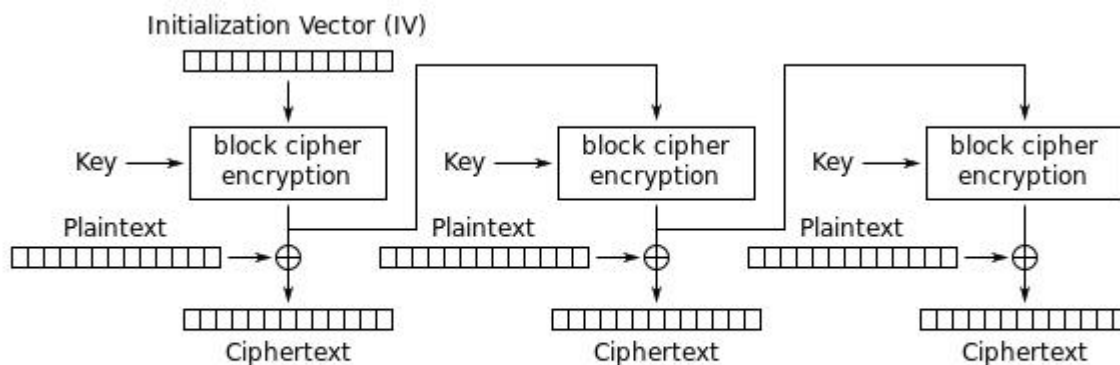
Electronic Codebook (ECB) mode encryption

두 번째로 CBC모드인데 앞서의 ECB에서의 암호화한 블록의 결과를 다음의 블록에 XOR 연산하여 나가는 게 특징이다. 이때 제일 처음의 암호화 시에는 마지막 블록 결과를 이용하거나 IV (Initial Vector)를 이용하게 된다.



Cipher Block Chaining (CBC) mode encryption

세 번째로 OFB 모드인데 IV를 암호화하여 그것을 다시 암호화한 후 계속 난수를 생성한다. 그렇게 생성된 난수 리스트를 XOR 연산에 의해 원문에 적용하여 암호화하는 방식이다. 즉, 블록 암호를 스트림 암호와 같이 사용한다고 보면 되겠다.



Output Feedback (OFB) mode encryption

- 평문의 각 블록은 XOR연산을 통해 이전 암호문과 연산되고 첫 번째 암호문에 대해서는 IV(Initial Vector)가 암호문 대신 사용된다. 이 때, IV는 제 2의 키가 될 수 있다.
- 암호문이 블록의 배수가 되기 때문에 복호화 후 평문을 얻기 위해서 Padding을 해야만 한다.

➤ openssl 라이브러리 제공 (AES 암호화 방식)

- 1) EVP 라이브러리
- 2) AES 라이브러리

2가지 라이브러리를 제공한다. EVP 라이브러리는 AES 기본 라이브러리를 provider pattern으로 감싼 것이다.

위키 암호화 운용 방식 : https://ko.wikipedia.org/wiki/%EB%B8%94%EB%A1%9D_%EC%95%94%ED%98%B8_%EC%9A%B4%EC%9A%A9_%EB%B0%A9%EC%8B%9D#.ED.8C.A8.EB.94.A9

➤ EVP 암호화 과정

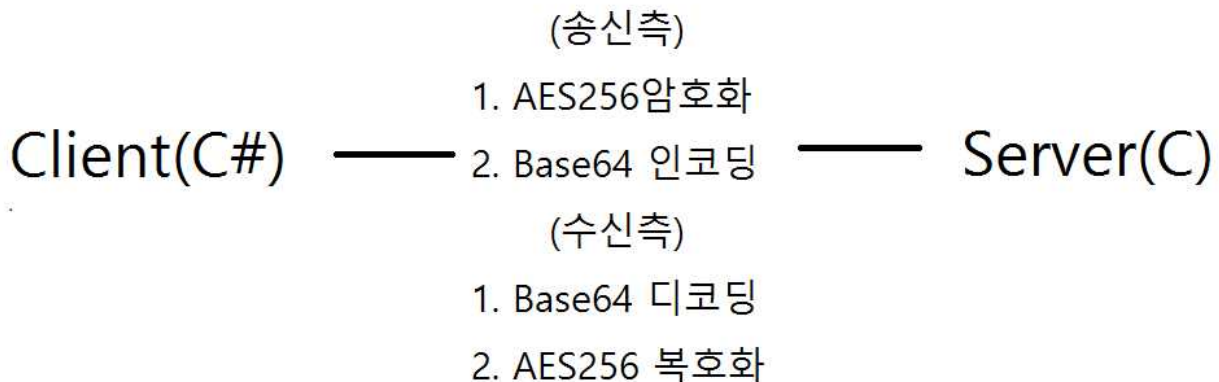
모든 암호화 라이브러리는 init, update, final 과정을 거친다.

- init: 암호화 관련 정보를 설정한다. (암호화 방식, 키 길이, 비밀 번호 등)
- update: 설정한 암호화 방식으로 블록을 분할해 한 블록씩 암호화를 수행한다.
- final: 입력한 plain text의 크기가 블록의 배수가 아닐 경우 데이터 끝에 여분의 데이터 바이트가 남게 되는 해당 바이트를 패딩하여 처리 가능한 크기의 블록으로 만든 다음 암호화를 수행한다.

출처 : <http://blog.naver.com/PostView.nhn?blogId=nextchoice&logNo=120125229265>

출처 : <http://jo.centis1504.net/?p=137>

➤ Client 와 Server 암호화, 복호화 과정



Server – C 언어

필요한 헤더

```
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
```

- 컴파일 시 -lssl, -lcrypto 옵션 필요

암호화 코드

```
#define BUF_LEN 128
void handleErrors(void)
{
    ERR_print_errors_fp(stderr);
    abort();
}

int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key, unsigned char
*iv, unsigned char *ciphertext)
{
    EVP_CIPHER_CTX *ctx;
    int len=0;
    int ciphertext_len;
    /* Create and initialise the context */
    if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors();
    /* Initialise the encryption operation. IMPORTANT - ensure you use a key
    * and IV size appropriate for your cipher
    * In this example we are using 256 bit AES (i.e. a 256 bit key). The
    * IV size for *most* modes is the same as the block size. For AES this
    * is 128 bits */
    if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
        handleErrors();
    /* Provide the message to be encrypted, and obtain the encrypted output.
    * EVP_EncryptUpdate can be called multiple times if necessary
    */

    if(1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len))
        handleErrors();
    ciphertext_len = len;
    /* Finalise the encryption. Further ciphertext bytes may be written at
    * this stage.
    */
    if(1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) handleErrors();
    ciphertext_len += len;
    /* Clean up */
    EVP_CIPHER_CTX_free(ctx);
    return ciphertext_len;
}
```

- 버퍼 길이는 CBC모드에선 16바이트씩 블록을 잡음
- 보내는 데이터의 길이는 알아서 16씩 잘라서 암호화하고 전송하므로 내가 전송하고 싶은 데이터 길이는 임의로 지정해도 될 것으로 보임
- 사용된 알고리즘은 AES256 _ CBC 모드를 사용하였음

Main 함수 내 변수 선언

```
/* A 256 bit key */
unsigned char *key = (unsigned char *)"01234567890123456789012345678901";

/* A 128 bit IV */
unsigned char *iv = (unsigned char *)"0123456789012345";
/* Buffer for ciphertext. Ensure the buffer is long enough for the
 * ciphertext which may be longer than the plaintext, dependant on the
 * algorithm and mode
 */
unsigned char send_ciphertext[MAXLINE]; //송신 할 암호화텍스트를 저장할 변수
int send_ciphertext_len; //송신 할 암호화텍스트의 크기
char* send_base64_cipher_text; //송신 할 인코딩(암호화(텍스트)) 변수
int send_base64_cipher_text_len; //송신 할 인코딩(암호화(텍스트)) 크기

unsigned char rcv_plaintext[MAXLINE]; // 디코딩(복호화)된 변수 = 평문
int rcv_plaintext_len; // 디코딩(복호화)된 크기 = 평문
char* rcv_ciphertext; // 수신된 암호화텍스트 디코딩한 변수
int rcv_ciphertext_len = 1; // 수신된 암호화텍스트 디코딩된 크기
/* Initialise the library */
ERR_load_crypto_strings();
OpenSSL_add_all_algorithms();
OPENSSL_config(NULL);
```

-rcv_ciphertext_len을 1로 설정해주어야한다. base64_decode()함수에서 0이 아닐 때 if문을 들어가서 값을 설정해주기 때문에 0이 아닌 가장 작은 값을 설정해 주어야 한다. 더불어, 숫자가 있을때만 보내지기 때문에 가장작은 숫자는 1이 된다.

Main 함수 내 Send 함수

```
//SEND 부분
if((fork_ret = fork()) > 0)
{
    // 부모 프로세스는 키보드 입력을 클라이언트로 송신

    while(fgets(sendline, MAXLINE, stdin)!=NULL)
    {
        sendline[strlen(sendline)] = '\0';
        size = strlen(sendline);
        printf("\n----- S E N D -----\n\n");
        printf("SEND DATA : %s\n\n",sendline);
        if(size!=0){
```

```

        send_ciphertext_len = encrypt(sendline, strlen(sendline), key, iv, send_ciphertext);
        send_ciphertext[send_ciphertext_len] = '\0';

        printf("AES256 ( SEND DATA ) : %s \n\n",send_ciphertext);
        send_base64_cipher_text = base64_encode((unsigned char *)send_ciphertext,
                                                send_ciphertext_len,&send_base64_cipher_text_len);
        printf("BASE64ENCODING ( AES256 ( SEND DATA ) ) : %s \n\n",send_base64_cipher_text);
        if(write(client_sock, send_base64_cipher_text, send_base64_cipher_text_len) == size)
        {
            printf("Error in write. \n");
        }
    }
}
}

```

- fgets() 함수로 데이터를 키보드로 입력받는다.
- encrypt() 함수를 통해 이미 정해져있던 key와 iv 값으로 입력받은 데이터를 암호화 한다.
- 암호화 된 데이터를 base64_encode() 함수를 통해 인코딩 한다.
- 그 후 write() 함수를 통해 데이터를 전송한다.

Main 함수 내 Receive 함수

```

//RECEIVE 부분
else if(fork_ret == 0)
{
    // 자식 프로세스는 클라이언트로부터 수신된 메시지를 화면에 출력
    while(1)
    {
        //recvline 초기화 해주어야 다음에 내용의 추가 쓰기가 되지 않는다.
        memset(recvline,0x00,MAXLINE);
        if((size = read(client_sock, recvline, MAXLINE)) < 0)
        {
            printf("Error if read. \n");
            close(client_sock);
            exit(0);
        }
        //평문에 섞여 들어가게되면 이상한 값이 추가되어 출력된다.
        memset(recv_plaintext,0x00,MAXLINE);
        printf("----- R E C E I V E -----\n\n");
        // 암호화&인코딩 된 값 출력
        printf("BASE64ENCODING ( AES256 ( RECEIVE DATA ) ) : %s\n\n", recvline);
        recv_ciphertext = base64_decode((unsigned char *)recvline, strlen(recvline),&recv_ciphertext_len);
        printf(" AES256 ( RECEIVE DATA ) : %s\n\n", recv_ciphertext); // 화면 출력
        recv_plaintext_len = decrypt(recv_ciphertext, recv_ciphertext_len, key, iv, recv_plaintext);
        recv_plaintext[recv_plaintext_len] = '\0';
        printf("RECEIVE DATA : %s\n\n", recv_plaintext); // 화면 출력
    }
}

```

```
}  
}
```

- read() 함수를 통해 데이터를 수신받는다.
- base64_decode() 함수를 통해 입력받는 데이터를 디코딩한다.
- decrypt() 함수를 통해 디코딩된 데이터를 복호화한다. 출력으로는 보낸 데이터가 나온다.

Client - c#

필요한 헤더

```
using System.Security.Cryptography;
```

암호화 복호화 함수

```
//AES 암호화 부분  
public String AES_encrypt(String Input, String key)  
{  
    RijndaelManaged aes = new RijndaelManaged();  
    aes.KeySize = 256;  
    aes.BlockSize = 128;  
    aes.Mode = CipherMode.CBC;  
    aes.Padding = PaddingMode.PKCS7;  
    aes.Key = Encoding.UTF8.GetBytes(key);  
    aes.IV = Encoding.UTF8.GetBytes("0123456789012345");  
    var encrypt = aes.CreateEncryptor(aes.Key, aes.IV);  
    byte[] xBuff = null;  
    using (var ms = new MemoryStream())  
    {  
        using (var cs = new CryptoStream(ms, encrypt, CryptoStreamMode.Write))  
        {  
            byte[] xXml = Encoding.UTF8.GetBytes(Input);  
            cs.Write(xXml, 0, xXml.Length);  
        }  
        xBuff = ms.ToArray();  
    }  
    String Output = Convert.ToBase64String(xBuff);  
    return Output;  
}  
public String AES_decrypt(String Input, String key)  
{  
    RijndaelManaged aes = new RijndaelManaged();  
    aes.KeySize = 256;  
    aes.BlockSize = 128;  
    aes.Mode = CipherMode.CBC;  
    aes.Padding = PaddingMode.PKCS7;
```

```

aes.Key = Encoding.UTF8.GetBytes(key);
aes.IV = Encoding.UTF8.GetBytes("0123456789012345");
var decrypt = aes.CreateDecryptor();
byte[] xBuff = null;
using (var ms = new MemoryStream())
{
    Console.WriteLine(Input);
    using (var cs = new CryptoStream(ms, decrypt, CryptoStreamMode.Write))
    {
        byte[] xXml = Convert.FromBase64String(Input);
        cs.Write(xXml, 0, xXml.Length);
    }
    xBuff = ms.ToArray();
}
String Output = Encoding.UTF8.GetString(xBuff);
return Output;
}
}

```

- 키 사이즈를 256으로 BlockSize는 128 모드는 CBC 패딩은 PKCS7으로 정의한다.
- 키와 IV값은 C와 동일한 값으로 설정해준다.

Client(C#) Receive() 함수

```

private void OnReceiveCallBack(IAsyncResult IAR)
{
    try
    {
        Socket tempSock = (Socket)IAR.AsyncState;
        int nReadSize = tempSock.EndReceive(IAR);
        if (nReadSize != 0)
        {
            tbDebug.Text += "WrWnWrWn----- R E C E I V E -----";
            //버퍼의 나머지 부분이 \0로 채워지기 때문에 trim()함수를 사용하여 \0를 제거해
            야한다. 그렇지 않다면 복호화가 되지않는다.(패딩의 길이 때문에)
            recvString = Encoding.UTF8.GetString(recvBuffer).Trim('\0');
            Array.Clear(recvBuffer, 0, recvBuffer.Length);
            tbDebug.Text += "WrWnWrWnBASE64ENCODING ( AES256 ( RECEIVE DATA ) ) : "
+ recvString;

            string chipper = AES_decrypt(recvString, "01234567890123456789012345678901");
            this.tbDebug.Text += "WrWnWrWnRECEIVE DATA : " + chipper;
        }
        this.Receive();
    }
    catch (SocketException se)
    {

```



```

        if (se.SocketErrorCode == SocketError.ConnectionReset)
        {
            this.BeginConnect();
        }
    }
}

```

- 수신한 데이터를 trim함수를 이용해 잘라야 한다. 그렇지 않다면 뒤에 'w0'부분으로 채워서 오기 때문에 복호화할 때 제대로된 값을 복호화할수 없다.(패딩으로 인해 복호화자체가 되지 않는다.)
- 위쪽에서는 보이지 않지만 Base64 디코딩의 값은 decrypt() 함수안에 들어가있다. 결론적으로 base64디코딩 후 복호화를 진행한다.

Client(C#) Send() 함수

```

public void BeginSend(string message)
{
    tbDebug.Text += "WrWnWrWn----- S E N D -----";
    tbDebug.Text += "WrWnWrWnSEND DATA : " + message;
    try
    {
        /* 연결 성공시 */
        if (clientSock.Connected)
        {
            string chipper = AES_encrypt(message, "01234567890123456789012345678901");
            byte[] buffer = Encoding.UTF8.GetBytes(chipper);
            clientSock.BeginSend(buffer, 0, buffer.Length, SocketFlags.None, new
AsyncCallback(SendCallBack), chipper);
        }
    }
    catch (SocketException e)
    {
        tbDebug.Text = "WrWnWrWn전송 에러 : " + e.Message;
    }
}

```

- encrypt() 함수를 통해 데이터를 암호화한다. base64 인코딩은 encrypt() 함수안에 존재한다.

참고사항

How to install 'Openssl' in Linux

1. Openssl 다운로드

홈페이지 : <http://www.openssl.org/source/>

2. 압축 해제

➤ `gzip -cd openssl-1.X.XX.tar.gz | tar xvf -`

3. 옵션 설정

압축 푼 디렉토리로 이동하고,

- `./config`
- `./config --prefix=/usr/`

둘 중 하나 선택.. 위에 꺼는 디폴트 값인 `/usr/local/openssl` 에 컴파일 됨
Prefix를 주면 원하는 설치 경로를 지정할 수 있음

4. `make` (약 20-25분 소요 – 보드에서 실행할 경우)

- 마지막에 error 1개 발생할 수 있음. 사용하는데 지장 없습니다.

5. `make install` (1-2분 소요 – 보드에서 실행할 경우)

6. openssl version 확인

```
root@PS-L16:~/openssl# openssl version
OpenSSL 1.1.0c 10 Nov 2016
```

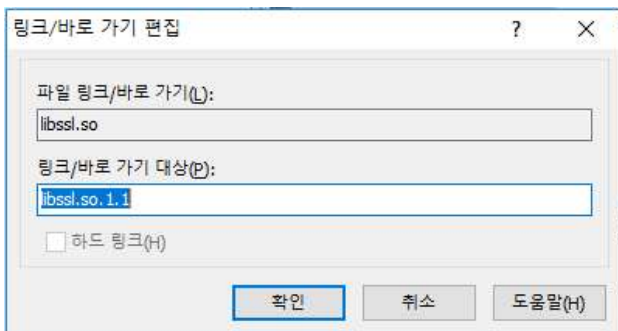
1. 참고 사항 중요. 필독.(From 서정원)

- Libssl.so.1.1 을 찾을 수 없다. Libcrypto.so.1.1을 찾을 수 없다. 는 메시지가 나올 때
 - /usr/lib로 가서 보면

libdbus-1.so.3	1 KB	2016-05-17 14:00	libstdc++.so.6	1 KB	2016-05-17 14:00
libcrypto.so.1.1	2,215 KB	2016-12-01 10:00	libstdc++.so	1 KB	2016-05-17 14:00
libcrypto.so	1 KB	2016-12-01 10:00	libstdc++.la	1 KB	2016-05-17 14:00
libcrypto.a	3,246 KB	2016-12-01 10:00	libssl.so.1.1	399 KB	2016-12-01 10:00
libcrypt.so	1 KB	2016-05-17 14:00	libssl.so.1.0.0	328 KB	2016-05-17 14:00
libcrypt.so	1 KB	2016-05-17 14:00	libssl.so	1 KB	2016-12-01 10:00
libcrypt.a	1 KB	2016-05-17 14:00	libssl.a	489 KB	2016-12-01 10:00
libcrypt.spec	1 KB	2016-05-17 14:00	libsanitizer.spec	1 KB	2016-05-17 14:00

볼 수 있을 것입니다. 이 화면은 제가 수정을 다 해 놓았기 때문에 정상 작동 할 것입니다.

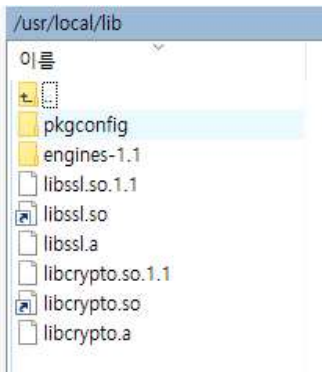
위와 같은 에러는 libcrypto.so 바로가기를 보시면 예전 버전으로 링크가 걸려 있을 것입니다. 이걸 최신 버전으로 연결해 줘야 합니다. libssl도 마찬가지 입니다.



이런 식으로 libssl.so.1.1 로 링크/바로가기 대상을 바꿔 주시면 됩니다.

- 만약 /usr/lib 에 최신 라이브러리 파일(링크 걸어줄 파일)이 없다면?

./config에 프리픽스를 설정한 곳을 찾아 보시길 바랍니다. 예를 들면 /usr/local/lib/ 에 위치해 있을 수 있습니다.



처럼 보이신다면 libssl.so.1.1 을 복사해서 /usr/lib에 넣으셔야 합니다.

- 컴파일 시 reference 오류, undeclared function error 등등 함수 참조 오류
 - 컴파일 -lssl -lcrypto 옵션 추가했는지 확인

컴파일 시 많은 warning이 발생하는데 하나하나 확인해 보기 어려웠습니다. 아마 api 내부에서 사용되는 함수들에 대한 컴파일 warning 인 것 같습니다.

- 컴파일 명령어

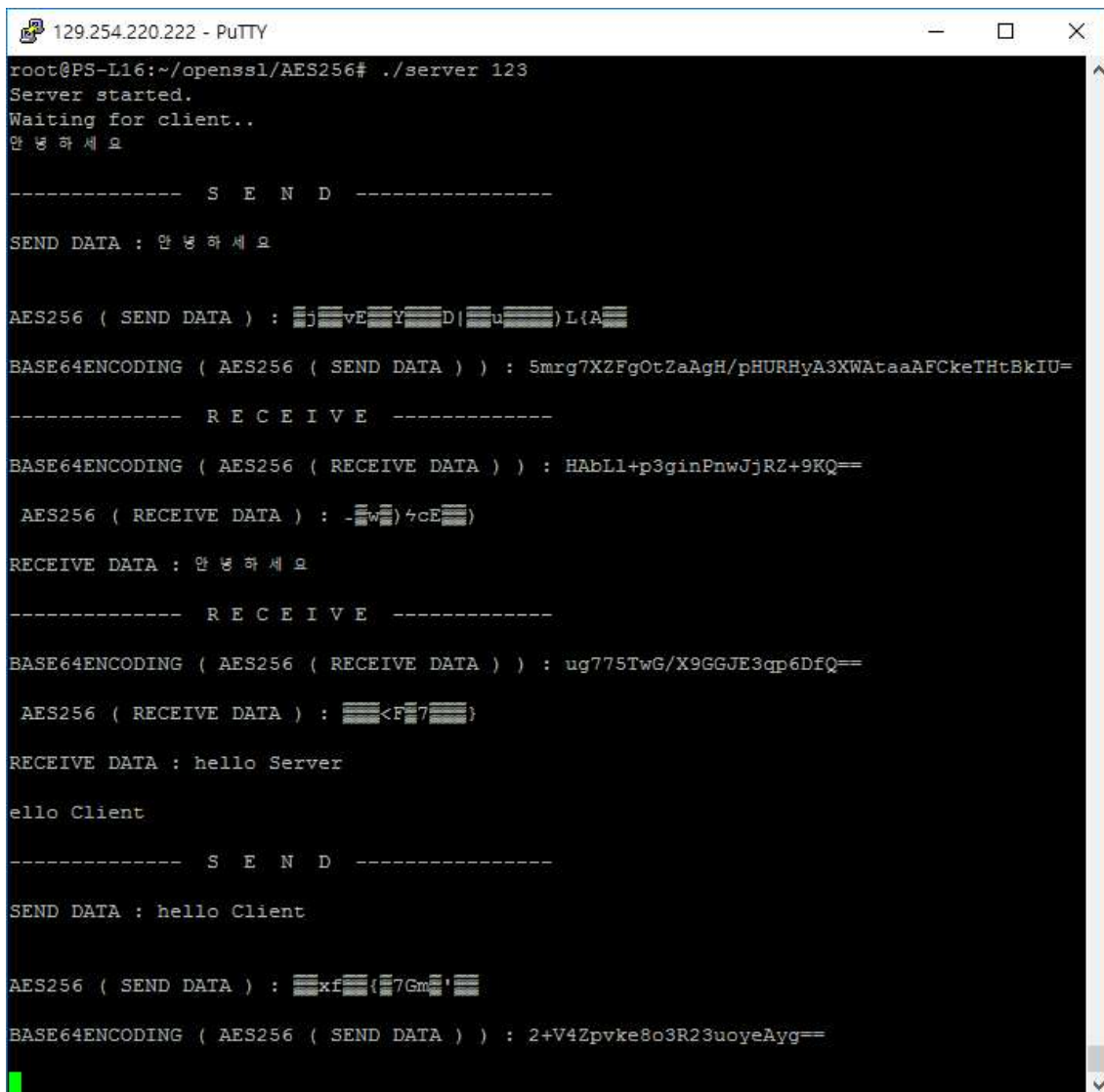
gcc -o server server.c -lcrypto -lssl

- 서버 실행

./server 123

- C언어 쪽 base64는 직접 코드로 구현이 되어있지만 openssl에서 제공하는 라이브러리가 있다고 알고있음
지금의 base64도 제대로 동작하지만 openssl을 사용하는것도 나쁘지 않다고 생각.

- Server쪽의 Screenshot



```
129.254.220.222 - PuTTY
root@PS-L16:~/openssl/AES256# ./server 123
Server started.
Waiting for client..
안녕하세요

----- S E N D -----
SEND DATA : 안녕하세요

AES256 ( SEND DATA ) : jvEYD|uL{A
BASE64ENCODING ( AES256 ( SEND DATA ) ) : 5mrg7XZFgOtZaAgH/pHURHyA3XWAtaaAFCkeTHtBkIU=
----- R E C E I V E -----
BASE64ENCODING ( AES256 ( RECEIVE DATA ) ) : HAbLl+p3ginPnwJjRZ+9KQ==
AES256 ( RECEIVE DATA ) : -w}4cE}
RECEIVE DATA : 안녕하세요

----- R E C E I V E -----
BASE64ENCODING ( AES256 ( RECEIVE DATA ) ) : ug775TwG/X9GGJE3qp6DfQ==
AES256 ( RECEIVE DATA ) : <F7}
RECEIVE DATA : hello Server
ello Client

----- S E N D -----
SEND DATA : hello Client

AES256 ( SEND DATA ) : xf{7Gm'
BASE64ENCODING ( AES256 ( SEND DATA ) ) : 2+V4Zpvke8o3R23uoyeAyg==
```

● Client쪽 스크린샷

