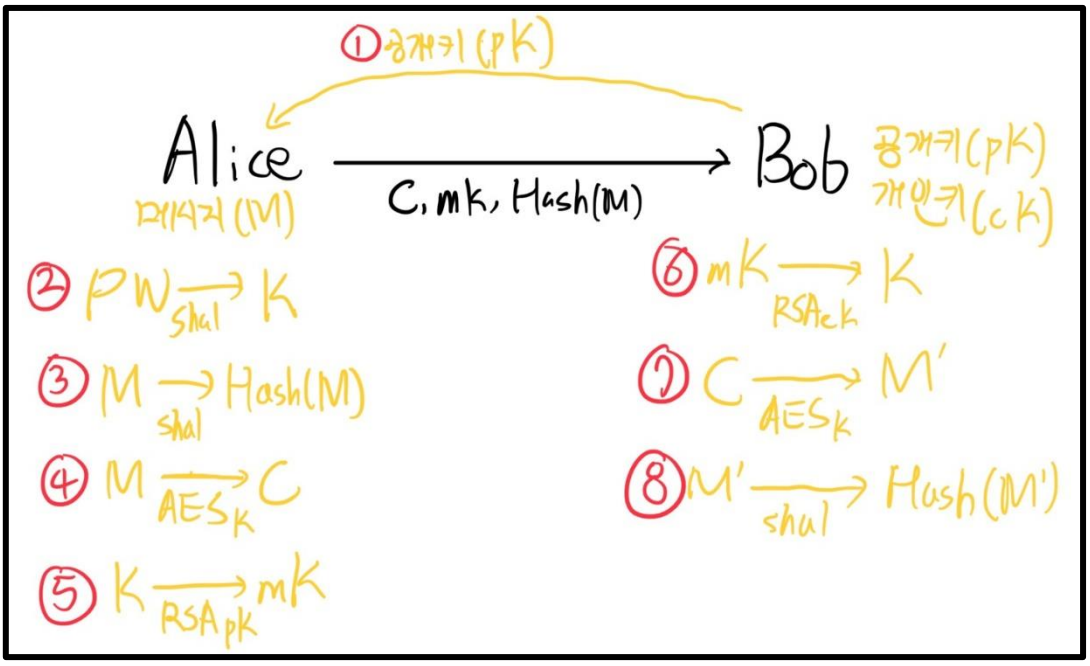
보안프로토콜 최종 프로젝트 보고서-20192233 박진철

1. 작동 순서



Alice가 Bob에게 파일을 보내주려고 함

1)Bob은 자신의 공개키와 개인키를 만들어 Alice에게 공개키를 보내줌

2)Alice는 자신이 입력한 패스워드를 sha1을 이용하여 AES 대칭키를 만듬

3)보내려는 파일의 해시값을 찾아냄

4)파일을 AES CBC모드로 암호화

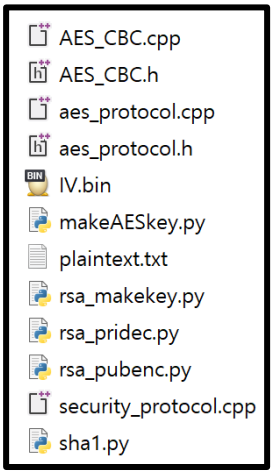
5)AES 대칭키를 Bob의 공개키를 이용하여 RSA로 암호화

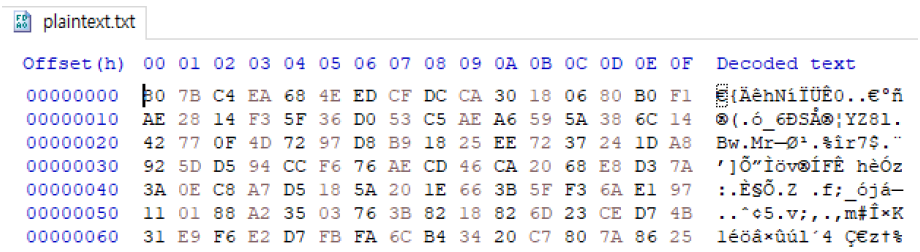
6)Bob은 Alice에게 받은 키를 자신의 개인키를 이용하여 RSA로 복호화

7)받은 암호화 파일을 AES CBC모드로 복호화

8)복호화한 파일의 해시값과 받은 해시값을 비교하여 이상이 없는지 확인

2. 구현에 쓰인 파일





-AES_CBC : AES CBC로 파일을 암호화하기 위한 알고리즘

-aes_protocol : AES에 필요한 알고리즘

-IV.bin : CBC모드에서 쓰이는 초기 벡터

-makeAESkey : AES 대칭키를 만드는 파일

-plaintext : 보내려고 하는 파일

-rsa_makekey : rsa에서 쓰일 공개키와 개인키를 만드는 파일

-rsa_pridec : rsa의 개인키를 이용하여 복호화하는 파일

- rsa_pubenc: rsa의 공개키를 이용하여 암호화하는 파일

-security_protocol : AES의 암/복호화를 해주는 파일

-sha1 : 파일의 해시값을 알려주는 파일

직접 구현한 파일 : AES_CBC , aes_protocol, security_protocol

Sha1 과 RSA 는 파이썬의 모듈을 활용

3. 구현 과정

1)Bob은 rsa_makekey를 이용하여 개인키와 공개키를 만듬

rsa_makekey.py - C:₩Users₩Jin_Cheol₩Docume

File Edit Format Run Options Window Help

```
from Crypto.PublicKey import RSA

key=RSA.generate(1024)
file_out=open("private.pem",'wb+')
file_out.write(key.exportKey('PEM'))
file_out.close()

new=key.publickey()
file_output=open("public.pem",'wb')
file_output.write(new.exportKey('PEM'))
file_output.close()
```

<rsa_makekey.py>

-개인키는 private.pem, 공개키는 public.pem에 저장됨

private.pem

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	2D	2D	2D	2D	2D	42	45	47	49	4E	20	52	53	41	20	50	-----BEGIN RSA P
00000010	52	49	56	41	54	45	20	4B	45	59	2D	2D	2D	2D	2D	0A	RIVATE KEY-----.
00000020	4D	49	49	43	58	51	49	42	41	41	4B	42	67	51	44	67	MIICXQIBAAKBgQDg
00000030	6B	43	4C	4C	58	75	6C	4D	4D	61	68	34	76	34	38	34	kCLLXulMMah4v484
00000040	45	4B	77	63	4B	56	6C	71	4C	5A	54	45	51	6A	68	6C	EKwcKVlqLZTEQjhl
00000050	55	4E	58	55	36	46	72	62	70	7A	47	74	4D	6A	2F	78	UNXU6FrbpzGtMj/x
00000060	0A	52	4C	56	48	38	67	6A	39	77	56	61	47	42	64	36	.RLVH8gj9wVaGBd6
00000070	78	41	33	74	74	31	53	55	5A	57	43	66	30	62	2B	52	xA3tt1SUZWCf0b+R
00000080	6B	65	58	38	70	48	45	65	52	53	34	4E	30	39	7A	33	keX8pHEeRS4N09z3
00000090	4D	6B	6E	66	57	58	4F	57	59	62	6F	75	63	58	55	47	MknfWXOWYboucXUG
000000A0	61	0A	69	41	33	47	68	45	52	64	2B	50	4C	2B	75	48	a.iA3GhERd+PL+uH
000000B0	6A	33	50	79	4A	2F	76	6A	45	52	64	44	62	2F	51	55	j3PyJ/vjERdDb/QU
000000C0	57	75	4F	6C	6D	6E	68	4C	6B	2F	50	56	4D	77	58	39	WuOlmnhLk/PVMwX9
000000D0	39	2F	34	45	6D	66	54	54	45	63	41	77	49	44	41	51	9/4EmfTTEcAwIDAQ
000000E0	41	42	0A	41	6F	47	41	44	2B	5A	4A	2F	62	63	38	63	AB.AoGAD+ZJ/bc8c
000000F0	6D	39	7A	70	6C	2B	6F	62	73	65	65	46	56	4A	63	30	m9zpl+obseeFVJc0
00000100	39	61	4E	57	58	43	69	4C	65	6A	6D	79	45	55	6A	2B	9aNWXCiLejmyEUj+
00000110	77	32	31	52	2B	44	5A	47	53	61	4D	5A	55	6E	55	51	w21R+DZGSaMZUnUQ
00000120	58	6E	6A	0A	75	2B	66	61	4B	42	4E	55	69	6D	61	2F	Xnj.u+faKBNUima/
00000130	4A	39	30	50	68	64	41	35	6F	31	37	63	31	67	6F	69	J90PhdA5o17c1goi
00000140	53	67	59	49	48	44	51	56	6E	56	76	78	43	5A	33	45	SgYIHDQVnVvxCZ3E
00000150	6D	5A	52	6B	42	48	42	33	52	6A	48	37	6A	61	4B	32	mZRkBHB3RjH7jaK2

public.pem

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	2D	2D	2D	2D	2D	42	45	47	49	4E	20	50	55	42	4C	49	-----BEGIN PUBLI
00000010	43	20	4B	45	59	2D	2D	2D	2D	0A	4D	49	47	66	4D		C KEY-----.MIGfM
00000020	41	30	47	43	53	71	47	53	49	62	33	44	51	45	42	41	A0GCSqGSIb3DQEBA
00000030	51	55	41	41	34	47	4E	41	44	43	42	69	51	4B	42	67	QUAA4GNADCBiQKBg
00000040	51	44	67	6B	43	4C	4C	58	75	6C	4D	4D	61	68	34	76	QDgkCLLXulMMah4v
00000050	34	38	34	45	4B	77	63	4B	56	6C	71	0A	4C	5A	54	45	484EKwcKVlq.LZTE
00000060	51	6A	68	6C	55	4E	58	55	36	46	72	62	70	7A	47	74	QjhlUNXU6FrbpzGt
00000070	4D	6A	2F	78	52	4C	56	48	38	67	6A	39	77	56	61	47	Mj/xRLVH8gj9wVaG
00000080	42	64	36	78	41	33	74	74	31	53	55	5A	57	43	66	30	Bd6xA3tt1SUZWCf0
00000090	62	2B	52	6B	65	58	38	70	48	45	65	52	0A	53	34	4E	b+RkeX8pHEeR.S4N
000000A0	30	39	7A	33	4D	6B	6E	66	57	58	4F	57	59	62	6F	75	09z3MknfWXOWYbou
000000B0	63	58	55	47	61	69	41	33	47	68	45	52	64	2B	50	4C	cXUGaiA3GhERd+PL
000000C0	2B	75	48	6A	33	50	79	4A	2F	76	6A	45	52	64	44	62	+uHj3PyJ/vjERdDb
000000D0	2F	51	55	57	75	4F	6C	6D	6E	68	4C	6B	2F	0A	50	56	/QUWuOlmnhLk/.PV
000000E0	4D	77	58	39	39	2F	34	45	6D	66	54	54	45	63	41	77	MwX99/4EmfTTEcAw
000000F0	49	44	41	51	41	42	0A	2D	2D	2D	2D	2D	45	4E	44	20	IDAQAB.-----END
00000100	50	55	42	4C	49	43	20	4B	45	59	2D	2D	2D	2D	2D		PUBLIC KEY-----

-Bob은 공개키인 public.pem을 Alice에게 전달

2)makeAESkey를 이용하여 AES 대칭키를 만들어냄

makeAESkey.py - C:₩Users₩Jin_Cheol₩Documents₩security_protocc

File Edit Format Run Options Window Help

```
import hashlib

HASH_NAME = "sha1"

txt = input("패스워드를 입력하세요: ")

text = txt.encode('utf-8')
sha1 = hashlib.new(HASH_NAME)
sha1.update(text)
result = sha1.hexdigest()
c=[]
for i in range(16):
    a=result[2*i:2*i+2]
    b=int(a,16)
    c.append(b)
keyname=input("키를 저장할 파일의 이름을 입력하세요: ")
with open(keyname,mode='w+b') as file:
    file.write(bytes(c))
```

<makeAESkey.py>

```
AESkey.py
패스워드를 입력하세요: kookmin
키를 저장할 파일의 이름을 입력하세요: key128.bin
>>>
```

key128.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	18	51	53	D9	8D	A2	F5	B2	3C	BA	6C	82	77	DA	1D	79

-kookmin을 입력하여 20바이트의 해시값을 얻어냄

-해시값을 16바이트만 남겨 AES128의 대칭키 암호로 사용

-키값은 key128.bin에 저장함

3)sha1을 이용하여 보내려는 파일의 해시값을 얻어냄

*sha1.py - C:₩Users₩Jin_Cheol₩Documents₩security_protocol₩security_p

File Edit Format Run Options Window Help

```
import hashlib

def sha1_for_largefile(filepath, blocksize=8192):
    sha_1 = hashlib.sha1()
    try:
        f = open(filepath, "rb")
    except IOError as e:
        print("file open error", e)
        return
    while True:
        buf = f.read(blocksize)
        if not buf:
            break
        sha_1.update(buf)
    return sha_1.hexdigest()

name=input("파일을 입력하세요 : ")
a=sha1_for_largefile(name)
keyname=input("해시값을 저장할 파일의 이름을 입력하세요: ")
with open(keyname,mode='w') as file:
    file.write(a)
```

<sha1.py>

```
파일을 입력하세요 : plaintext.txt
해시값을 저장할 파일의 이름을 입력하세요: phash.txt
>>>
```

phash.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

4ac17944db566fcfce4a24085a667a4ca70c2345

-보내려는 파일인 plaintext.txt의 해시값을 phash.txt에 저장

4)보내려는 파일 plaintext.txt를 security_protocol로 AES CBC 암호화

4-1)aes_protocol: AES에 필요한 함수들이 들어있음

```
typedef unsigned char byte; //바이트 단위로 가져옴

//subbytes, keyexpansion에 필요한 sbox 생성
byte Sbox[256] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};

//inverse subbytes에 필요한 Invsbox 생성
byte InvSbox[256] = {
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
};

//keyexpansion에 필요한 rcon 생성
byte rcon[10] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36 };
```

<sbox와 rcon>

```
//유한체 GF(2^8)에서의 곱셈을 위한 xbox함수
//x^8 mod m(x) = x^4 + x^3 + x + 1임을 이용
byte Xtime(byte fx)
{
    return (((fx >> 7) & 0x01) == 1 ? (fx << 1) ^ 0x1b : fx << 1);
    /*
    f(x)에 x^7이 존재한다면, x를 곱해 x^8을 만든 후 x^8을 x^4 + x^3 + x + 1로 바꿈
    f(x)에 x^7이 존재하지 않는다면, x를 곱하기만 함
    */
}

//유한체 GF(2^8)에서의 곱셈
byte Mul(byte fx, byte gx)
{
    byte z; // z = f(x) * g(x)
    int coef; // 계수 (0 또는 1)
    z = 0x00;
    //f(x)=a7(x^7)+a6(x^6)+...+a0
    for (int i = 7; i >= 0; i--) {
        coef = (fx >> i) & 0x01; // a7, a6, a5, ... ,a0
        /*
        f(x)*g(x)
        = g(x)*a0 + g(x)*a1*x + g(x)*a2*x^2 + g(x)*a3*x^3 + ... + g(x)*a7*x^7
        = g(x)*a0 + x*( g(x)*a1 + g(x)*a2*x + g(x)*a3*x^2 + ... +g(x)*a7*x^6 )
        = g(x)*a0 + x*( g(x)*a1 + x*( g(x)*a2 + g(x)*a3*x + ... +g(x)*a7*x^5)  )
        = g(x)*a0 + x*( ...   + x*(g(x)*a7) )  )
        */
        z = Xtime(z);
        if (coef == 1) {
            z = z ^ gx;
        }
    }
    return z;
}
```

<유한체에서의 곱셈>

```
//addroundkey 단계
void AddRoundKey(byte* state, byte* roundkey)   //입력값, 라운드키를 입력 받음
{
    for (int i = 0; i < 16; i++) {
        state[i] ^= roundkey[i];    //블럭에 라운드 키를 XOR함
    }
}
```

```
//subbytes 단계
void SubByte(byte* state)
{
    for (int i = 0; i < 16; i++) {
        state[i] = Sbox[state[i]];  //블럭 안의 바이트를 sbox의 값으로 치환
    }
}

//inverse subbytes 단계
void InvSubByte(byte* state)
{
    for (int i = 0; i < 16; i++) {
        state[i] = InvSbox[state[i]];   //블럭 안의 바이트를 Invsbox의 값으로 치환
    }
}
```

```
//shiftrows 단계
void ShiftRow(byte* state)
{
    byte x[16];

    /*
    state[i]=si(0<=i<15)
    s0  s4  s8  s12  => s0  s4  s8  s12
    s1  s5  s9  s13     s5  s9  s13 s1
    s2  s6  s10 s14     s10 s14 s2  s6
    s3  s7  s11 s15     s15 s3  s7  s11
    두번째 줄은 왼쪽으로 한 칸, 세번째 줄은 두 칸, 네번째 줄은 세 칸 만큼 밈
    */

    x[0] = state[0];
    x[1] = state[5];
    x[2] = state[10];
    x[3] = state[15];

    x[4] = state[4];
    x[5] = state[9];
    x[6] = state[14];
    x[7] = state[3];

    x[8] = state[8];
    x[9] = state[13];
    x[10] = state[2];
    x[11] = state[7];

    x[12] = state[12];
    x[13] = state[1];
    x[14] = state[6];
    x[15] = state[11];
    //x에 shiftrow한 배열을 받음

    for (int i = 0; i < 16; i++) {
        state[i] = x[i];    //입력값에 x를 넣음
    }
}
```

```
//inverse shiftrows 단계
void InvShiftRow(byte* state)
{
    byte x[16];

    /*
    state[i]=si(0<=i<15)
    s0  s4  s8  s12  => s0  s4  s8  s12
    s1  s5  s9  s13     s13 s1  s5  s9
    s2  s6  s10 s14     s10 s14 s2  s6
    s3  s7  s11 s15     s7  s11 s15 s3
    shiftrow 단계와 반대로
    두 번째 줄은 왼쪽으로 3번, 세 번째 줄은 2번, 네 번째 줄은 1번 밈
    */

    x[0] = state[0];
    x[1] = state[13];
    x[2] = state[10];
    x[3] = state[7];

    x[4] = state[4];
    x[5] = state[1];
    x[6] = state[14];
    x[7] = state[11];

    x[8] = state[8];
    x[9] = state[5];
    x[10] = state[2];
    x[11] = state[15];

    x[12] = state[12];
    x[13] = state[9];
    x[14] = state[6];
    x[15] = state[3];

    for (int i = 0; i < 16; i++) {
        state[i] = x[i];
    }
}
```

```
//mixcolumns 단계
void MixColumn(byte* state)
{
    byte MixCol[4][4] = { {0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01},
                          {0x01, 0x01, 0x02, 0x03} , {0x03, 0x01, 0x01, 0x02} };
    byte x[4][4];
    byte temp;

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            x[i][j] = state[i * 4 + j];
        }
    } //입력받은 state를 4x4 행렬로 바꿈
    /*
    state[i]=si(0<=i<15)
    s0  s4  s8  s12
    s1  s5  s9  s13
    s2  s6  s10 s14
    s3  s7  s11 s15
    */

    for (int i = 0; i < 4; i++) {
        /* state[i]=s[i], MixCol[i][j]=m[i][j]
        s[i * 4 + j] += m[j][k] * x[i][k](k=0~k=3)
        */
        for (int j = 0; j < 4; j++) {
            temp = 0x00; // temp=s[i * 4 + j]
            for (int k = 0; k < 4; k++) {
                temp ^= Mul(x[i][k], MixCol[j][k]);
                //temp에 m[j][k]과 x[i][k]를 곱한값들을 합함(XOR)
            }
            state[i * 4 + j] = temp;
        }
    }
}
```

```
//inverse mixcolumns 단계
void InvMixColumn(byte* state)
{
    byte InvMixCol[4][4] = { {0x0e, 0x0b, 0x0d, 0x09}, {0x09, 0x0e, 0x0b, 0x0d},
                             {0x0d, 0x09, 0x0e, 0x0b} , {0x0b, 0x0d, 0x09, 0x0e}};

    byte x[4][4];
    byte temp;

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            x[i][j] = state[i * 4 + j];
        }
    }

    for (int i = 0; i < 4; i++) {
        /* state[i]=s[i], InvMixCol[i][j]=im[i][j]
        s[i * 4 + j] += im[j][k] * x[i][k](k=0~k=3)
        */
        for (int j = 0; j < 4; j++) {
            temp = 0x00; // temp=s[i * 4 + j]
            for (int k = 0; k < 4; k++) {
                temp ^= Mul(x[i][k], InvMixCol[j][k]);
                //temp에 im[j][k]과 x[i][k]를 곱한값들을 합함(XOR)
            }
            state[i * 4 + j] = temp;
        }
    }
}
```

```
void KeySchedule(byte Key[16], byte rk[11][16])
{
    for (int i = 0; i < 16; i++) {
        rk[0][i] = Key[i];
    }

    byte x[4];

    for (int i = 1; i < 11; i++) {
        x[0] = rk[i-1][13];
        x[1] = rk[i - 1][14];
        x[2] = rk[i - 1][15];
        x[3] = rk[i - 1][12];

        for (int j = 0; j < 4; j++) {
            x[j] = Sbox[x[j]];
        }

        rk[i][0] = rk[i - 1][0] ^ x[0] ^ rcon[i-1];
        rk[i][1] = rk[i - 1][1] ^ x[1];
        rk[i][2] = rk[i - 1][2] ^ x[2];
        rk[i][3] = rk[i - 1][3] ^ x[3];

        for (int j = 4; j < 16; j++)
        {
            rk[i][j] = rk[i - 1][j] ^ rk[i][j - 4];
        }
    }
}
```

```
void Dec_KeySchedule(byte Key[16], byte rk[11][16])
{
    byte erk[11][16];

    KeySchedule(Key, erk);

    for (int i = 0; i < 11; i++) {
        for (int j = 0; j < 16; j++) {
            rk[i][j] = erk[10 - i][j];
        }
    }
}
```

<addroundkey, subbytes, shiftrow, mixcolumn, keyschedule>

```
//AES 암호화
void AES_Encrypt(byte* PT, byte RK[11][16], byte* CT) //평문, 키, 결과값을 저장할 암호문을 입력받음
{
    byte state[16];

    //0라운드
    for (int i = 0; i < 16; i++) {
        state[i] = PT[i]; //state에 평문을 입력받음
    }

    AddRoundKey(state, RK[0]);
    //0라운드: 라운드 키 중 첫번째 키를 사용, addroundkey를 한번 실행

    for (int i = 1; i <= 9; i++) { //1-9라운드
        SubByte(state);
        ShiftRow(state);
        MixColumn(state);
        AddRoundKey(state, RK[i]);
        //1-9라운드: subbytes, shiftrows, mixcolumns, addroundkey를 순서대로 실행
    }

    //10라운드
    SubByte(state);
    ShiftRow(state);
    AddRoundKey(state, RK[10]);
    //subbytes, shiftrows, addroundkey를 순서대로 실행

    for (int i = 0; i < 16; i++) {
        CT[i] = state[i]; //결과 값인 암호문 출력
    }
}
```

```
//AES 복호화
void AES_Decrypt(byte* CT, byte RK[11][16], byte* PT) //암호문, 키, 결과값을 저장할 평문을 입력받음
{
    byte state[16];

    //0라운드
    for (int i = 0; i < 16; i++) {
        state[i] = CT[i]; //state에 암호문을 입력받음
    }
    AddRoundKey(state, RK[10]);

    for (int i = 9; i > 0; i--) { //1-9라운드
        InvShiftRow(state);
        InvSubByte(state);
        AddRoundKey(state, RK[i]);
        InvMixColumn(state);
        //inverse shiftrows, inverse subbytes, addroundkey, inverse mixcolumns를 순서대로 실행
    }

    //10라운드
    InvShiftRow(state);
    InvSubByte(state);
    //암호화의 0라운드에서 쓰인 라운드키는 원래 키값이므로 키값 그대로를 받음
    AddRoundKey(state, RK[0]);
    //inverse shiftrows, inverse subbytes, addroundkey를 순서대로 실행

    for (int i = 0; i < 16; i++) {
        PT[i] = state[i]; //결과값인 평문 출력
    }
}
```

<AES 암호화와 복호화>

4-2)AES_CBC: AES를 CBC모드로 구현

```
//파일의 길이를 알려주는 함수
int FileSize(const char* fname)
{
    FILE* myfile = fopen(fname, "rb");
    int file_length; //file_length에 파일의 길이를 저장

    fseek(myfile, 0, SEEK_END);
    file_length = ftell(myfile);
    fclose(myfile);

    return file_length; //file_length 반환
}
```

<파일의 길이를 반환하는 FileSize>

```
//파일을 패딩하는 함수
void File_Padding(const char* input, const char* output)
{
    FILE* infile = fopen(input, "rb");
    FILE* outfile = fopen(output, "wb");

    int file_len;
    file_len = FileSize(input);

    int num_blocks = file_len / 16 + 1;
    int remainder = file_len - (num_blocks - 1) * 16;

    byte nblock[16]; //패딩을 안해도 되는 블록(마지막 블록 전까지)

    for (int i = 0; i < num_blocks - 1; i++) {
        fread((char*)nblock, sizeof(byte), 16, infile);
        fwrite((char*)nblock, sizeof(byte), 16, outfile);
    }//마지막 블록 전까지는 output파일에 옮겨줌

    byte last_block[16]; //마지막 블록
    byte ch;

    if (remainder == 0) {
        fclose(infile);
        fclose(outfile);
    }//마지막 블록이 없다면 그대로 끝냄
    else { //패딩할 블록이 있다면
        for (int i = 0; i < remainder; i++) {
            fread((char*)&ch, sizeof(byte), 1, infile);
            last_block[i] = ch;
        } //남은 데이터는 옮겨줌

        byte padding_number = 0x00 + (16 - remainder);
        for (int i = remainder; i < 16; i++) {
            last_block[i] = padding_number; //빈공간은 남은 공간의 길이 숫자를 넣어서 패딩함
        }
        fwrite((char*)last_block, sizeof(byte), 16, outfile); //패딩까지 한 마지막 블록을 output파일에 옮겨줌

        fclose(infile);
        fclose(outfile);
    }
}
```

<파일을 패딩하는 File_Padding>

```
//패딩을 지우는 함수
void erase_padding(const char* pad_name, const char* origin_name)
{
    FILE* infile = fopen(pad_name, "rb");
    FILE* outfile = fopen(origin_name, "wb");

    int file_len;
    file_len = FileSize(pad_name);
    int num_blocks = file_len / 16 + 1;

    byte buffer[16];
    int x;

    for (int i = 0; i < num_blocks - 2; i++) {
        fread((char*)buffer, sizeof(byte), 16, infile);
        fwrite((char*)buffer, sizeof(byte), 16, outfile);
    }//패딩된 부분의 전 블록까지 origin파일에 저장

    byte last_block[16];
    fread((char*)last_block, sizeof(byte), 16, infile); //마지막 블록을 last_block에 저장
    for (int i = 0; i < 16; i++) {
        if (last_block[16 - i] == i) {
            for (int j = 16 - i; j < 16; j++) {
                if (last_block[j] == i) x = i; //마지막 블록에서 숫자 i가 마지막에서 i개 만큼 존재한다면 x=i
                else x = 0;
            }
            if (x == i) break;
        }
        else x = 0;
    }
    for (int i = 0; i < 16 - x; i++) { //0부터 16-x까지의 부분을 origin파일에 저장
        fwrite((char*)&last_block[i], sizeof(byte), 1, outfile);
    }

    fclose(infile);
    fclose(outfile);
}
```

<패딩된부분을 지우는 erase_padding>

```c
//CBC모드 AES 파일 암호화
void file_AES_CBC(const char *pt_name, byte IV[16], byte Key[16], const char *ct_name)
{
    byte rk[11][16];
    FILE *pt_file;
    FILE *ct_file;

    byte pt_block[16];
    byte ct_block[16];

    KeySchedule(Key, rk); //라운드키 획득

    const char* pad_file = "AES_CBC-Padded.bin";
    File_Padding(pt_name, pad_file); //파일을 패딩하여 저장

    int file_len;
    file_len = FileSize(pad_file);
    int num_blocks = file_len / 16 + 1;

    pt_file = fopen(pad_file, "rb");
    ct_file = fopen(ct_name, "wb");

    for (int i = 0; i < num_blocks - 1; i++) {
        if (i == 0) { //CBC 첫블록
            for (int j = 0; j < 16; j++) {
                fread((char*)&pt_block[j], sizeof(byte), 1, pt_file); //CBC의 첫블록을 pt_block에 저장
                pt_block[j] = pt_block[j] ^ IV[j]; //pt_block과 초기벡터를 XOR연산 수행
            }
            AES_Encrypt(pt_block, rk, ct_block); //pt_block를 암호화하여 ct_block에 1블록 입력
            fwrite((char*)ct_block, sizeof(byte), 16, ct_file); //암호화파일에 ct_block을 1블록 입력
        }
        else { //첫블록 이후
            for (int j = 0; j < 16; j++) {
                fread((char*)&pt_block[j], sizeof(byte), 1, pt_file); //pt_block에 평문파일을 1블록씩 입력
                pt_block[j] = pt_block[j] ^ ct_block[j]; //pt_block과 직전의 암호문 블록을 XOR연산 수행
            }
            AES_Encrypt(pt_block, rk, ct_block); //pt_block를 암호화하여 ct_block에 1블록 입력
            fwrite((char*)ct_block, sizeof(byte), 16, ct_file); //암호화파일에 ct_block을 1블록씩 입력
        }
    }

    fclose(pt_file);
    remove(pad_file); //패딩되었던 파일은 삭제
    fclose(ct_file);
}
```

<CBC모드를 통한 파일 AES 암호화>

```c
//CBC모드 AES 파일 복호화
void Inv_file_AES_CBC(const char *ct_name, byte IV[16], byte Key[16], const char *pt_name)
{
    byte rk[11][16];
    FILE *pt_file;
    FILE *ct_file;

    byte pt_block[16];
    byte ct1_block[16];
    byte ct2_block[16];

    KeySchedule(Key, rk); //라운드키 획득

    const char* pad_file = "Inv_AES_CBC-Padded.bin"; //패딩을 지우기 전인 파일

    int file_len;
    file_len = FileSize(ct_name);
    int num_blocks = file_len / 16 + 1;

    ct_file = fopen(ct_name, "rb");
    pt_file = fopen(pad_file, "wb");

    for (int i = 0; i < num_blocks - 1; i++) {
        if (i == 0) { //CBC 첫블록
            fread((char*)ct1_block, sizeof(byte), 16, ct_file); //암호문파일의 첫블록을 ct1_block에 저장
            AES_Decrypt(ct1_block, rk, pt_block); //ct1_block을 복호화하여 pt_block에 평문 1블록 입력
            for (int j = 0; j < 16; j++) {
                pt_block[j] = pt_block[j] ^ IV[j]; //pt_block과 초기벡터를 XOR연산 수행
                fwrite((char*)&pt_block[j], sizeof(byte), 1, pt_file); //pad_file에 pt_block 1블록 저장
            }
        }
        else { //첫블록 이후
            fread((char*)ct2_block, sizeof(byte), 16, ct_file); //ct2_block에 암호문파일을 1블록씩 입력
            AES_Decrypt(ct2_block, rk, pt_block); //ct2_block을 복호화하여 pt_block에 1블록 입력
            for (int j = 0; j < 16; j++) {
                pt_block[j] = pt_block[j] ^ ct1_block[j]; //pt_block과 직전의 암호문 1블록을 XOR연산 수행
                ct1_block[j] = ct2_block[j]; //ct1_block에 암호문 1블록을 저장
                fwrite((char*)&pt_block[j], sizeof(byte), 1, pt_file); //pad_file에 패딩이 포함된 평문을 입력
            }
        }
    }

    fseek(pt_file, 0, SEEK_SET);
    erase_padding(pad_file, pt_name); //pad_file의 패딩부분을 제거하여 pt파일에 넣음

    fclose(pt_file);
    remove(pad_file);
    fclose(ct_file); // 파일 닫기
}
```

<CBC모드를 통한 파일 AES 암호화>

4-3)security_protocol: 위의 두 소스코드를 이용한 파일 암호화/복호화 프로그램

```
//파일을 암호화하는 함수
void se_encrypt(const char* pt_name, const char* Key_name, const char* IV_name, const char* ct_name)
{
    FILE* keyfile;
    FILE* IVfile;
    byte CBCkey[16];
    byte IV[16];

    keyfile = fopen(Key_name, "rb");
    IVfile = fopen(IV_name, "rb");

    fread((char*)CBCkey, sizeof(byte), 16, keyfile);
    fread((char*)IV, sizeof(byte), 16, IVfile);
    file_AES_CBC(pt_name, IV, CBCkey, ct_name);


}

//파일을 복호화하는 함수
void se_decrypt(const char* ct_name, const char* Key_name, const char* IV_name, const char* pt_name)
{
    FILE* keyfile;
    FILE* IVfile;
    byte CBCkey[16];
    byte IV[16];

    keyfile = fopen(Key_name, "rb");
    IVfile = fopen(IV_name, "rb");

    fread((char*)CBCkey, sizeof(byte), 16, keyfile);
    fread((char*)IV, sizeof(byte), 16, IVfile);
    Inv_file_AES_CBC(ct_name, IV, CBCkey, pt_name);
}
```

<파일을 암호(복호)화 하는 함수>

```
int main()
{
    int w_check; //원하는 작업을 입력받음
    char pt_filename[80]; //평문 파일
    char ct_filename[80]; //암호된 파일
    char Key_filename[80]; //키값이 저장되어있는 파일
    char IV_filename[80]; //초기벡터가 저장되어있는 파일

    //암호화된 파일에 올바른 확장자명을 입력받기 위해 쓰는 변수
    char drive[11];
    char dir[256];
    char fname[256];
    char ext[256];
    //변수에 작업을 수행할 파일의 정보를 입력받음

    printf("₩n원하는 작업을 입력하시오.(1: 파일 암호화, 2: 파일 복호화):");
    scanf("%d", &w_check);
    if (w_check == 1) { //파일 암호화
        printf("암호화할 평문파일 이름 입력:");
        scanf("%s", pt_filename);
        printf("암호화 키 입력:");
        scanf("%s", Key_filename);
        printf("CBC모드를 수행하기 위한 초기벡터 입력:");
        scanf("%s", IV_filename);
        char ct_name[80];
        printf("암호화파일 이름 입력:");
        scanf("%s", ct_name);
        _splitpath(pt_filename, drive, dir, fname, ext);
        sprintf(ct_filename, "%s%s", &ct_name, &ext); //암호화파일이름+평문의확장자명
        printf("[암호화 수행]₩n");
        se_encrypt(pt_filename, Key_filename, IV_filename, ct_filename);
        printf("%s로 파일 암호화 수행 완료", &ct_filename);
    }
    else { //파일 복호화
        printf("복호화할 암호문파일 이름 입력:");
        scanf("%s", ct_filename);
        printf("암호화 키 입력:");
        scanf("%s", Key_filename);
        printf("CBC모드를 수행하기 위한 초기벡터 입력:");
        scanf("%s", IV_filename);
        char pt_name[80];
        printf("복호화파일 이름 입력:");
        scanf("%s", pt_name);
        _splitpath(ct_filename, drive, dir, fname, ext); //복호화파일이름+암호문의확장자명
        sprintf(pt_filename, "%s%s", &pt_name, &ext);
        printf("[복호화 수행]₩n");
        se_decrypt(ct_filename, Key_filename, IV_filename, pt_filename);
        printf("%s로 파일 복호화 수행 완료", &pt_filename);
    }
}
```

<사용자에게 파일을 입력받아 작업을 수행>

IV.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	F7	24	55	8D	B3	43	3A	52	3F	4E	51	A5	BE	A7	04	97

```
원하는 작업을 입력하시오.(1: 파일 암호화, 2: 파일 복호화):1
암호화할 평문파일 이름 입력:plaintext.txt
암호화 키 입력:key128.bin
CBC모드를 수행하기 위한 초기벡터 입력:IV.bin
암호화파일 이름 입력:cipher
[암호화 수행]
cipher.txt로 파일 암호화 수행 완료
C:₩Users₩Jin_Cheol₩Documents₩security_protocol₩x64₩Debug₩secu
었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

-plaintext.txt를 key128.bin을 키값으로, IV.bin을 초기벡터로 설정하여 암호화 수행

cipher.txt

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	2F	F6	84	08	B6	D0	5C	8D	2D	10	78	31	82	B5	36	6F
00000010	9E	DA	EB	44	3A	B4	E6	45	CA	34	F3	0F	CF	44	A8	E8
00000020	92	81	8F	B0	4D	65	AE	A5	BB	EA	EC	11	5B	9C	9B	D1
00000030	7C	84	C7	17	CC	B2	50	75	A1	52	DD	5F	E1	D0	01	8F
00000040	3A	53	B7	06	5C	4A	DA	B0	F2	56	59	1F	E6	6A	B8	53
00000050	A6	FC	F8	18	F3	80	B2	FE	BA	A4	22	4D	3F	B7	0C	D8
00000060	B4	A1	00	23	55	FA	53	90	FB	0B	CC	C0	F0	9B	45	1C

-암호화된 파일은 cipher.txt로 저장됨

5) rsa_pubenc를 이용하여 대칭키를 Bob의 공개키로 암호화

rsa_pubenc.py - C:₩Users₩Jin_Cheol₩Documents₩security_protoc

File Edit Format Run Options Window Help

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

name=input("공개키로 암호화할 키 파일을 입력하세요: ")

with open(name,'r+b')as file:
    ade=file.read()

public_key=RSA.import_key(open('public.pem').read())
cipher=PKCS1_OAEP.new(public_key)

enc=cipher.encrypt(ade)
print(enc)

with open(name,'w+b')as file:
    file.write(enc)
```

<rsa_pubenc.py>

```
공개키로 암호화할 키 파일을 입력하세요: key128.bin
b'₩xe0(₩x90:₩xa2₩xb4XE5₩x10₩x88'₩x07₩xf1₩xb4₩xe0₩xdd₩xca!?A₩xd1₩xd50₩xc3₩₩89₩xd8
₩xac₩xb5₩xc2₩xccu₩x89₩xbe₩x1a₩x94₩₩x130₩xec₩xbe₩x91₩xa7₩xa9t₩x92₩xbf₩x8b₩xa79₩x
a0m0₩xfdC₩x08₩xdfH₩xdd₩xf7/t₩xe6V₩xea₩x00₩xca₩xc8₩xeb₩xc6E₩xfa₩xa5*!₩x84₩xe6₩xfa
₩xd9N₩xcf₩xa4₩x18r₩x9c₩xe7V₩xfd₩xdd₩x86₩xe8₩xc3₩xf6B₩xd6₩xcf₩xd7₩x98K₩x1e₩xad₩x0
5₩xb9`-₩x83₩x82₩xab₩x19₩xc6₩xeeWpV₩xe3₩x8c1₩xe1]₩x12$₩xb8y ₩x86'
>>>
```

-키값이 들어있는 key128.bin을 RSA암호화

key128.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	E0	28	90	3B	A2	B4	58	45	35	10	88	60	07	F1	B4	E0
00000010	DD	CA	6C	3F	41	D1	D5	30	C3	5C	38	39	D8	AC	B5	C2
00000020	CC	75	89	BE	1A	94	57	13	4F	EC	BE	91	A7	A9	74	92
00000030	BF	8B	78	A7	39	A0	6D	4F	FD	43	08	DF	48	DD	F7	2F
00000040	74	E6	56	EA	00	CA	C8	EB	C6	45	FA	A5	2A	21	84	E6
00000050	FA	D9	4E	CF	A4	18	72	9C	E7	56	FD	DD	86	E8	C3	F6
00000060	42	D6	CF	D7	98	4B	1E	AD	05	B9	60	2D	83	82	AB	19
00000070	C6	EE	57	70	56	E3	8C	31	E1	5D	12	24	B8	79	20	86

-key128.bin이 암호화되었음

Alice는 Bob에게 암호화된 파일(cipher.text), 초기벡터값(IV.bin), 암호화된 대칭키(key128.bin), 해시값(phash.txt)을 줌

6)Bob은 rsa_pridec를 이용하여 대칭키를 자신의 개인키로 복호화

rsa_pridec.py - C:₩Users₩Jin_Cheol₩Documents₩security_protocol

File Edit Format Run Options Window Help

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

name=input("개인키로 복호화할 키 파일을 입력하세요: ")

with open(name,'r+b')as file:
    ade=file.read()

private_key=RSA.import_key(open('private.pem').read())
cipher=PKCS1_OAEP.new(private_key)

dec=cipher.decrypt(ade)
print(dec)

with open(name,'w+b')as file:
    file.write(dec)
```

<rsa_pridec.py>

```
개인키로 복호화할 키 파일을 입력하세요: key128.bin
b'₩x18QS₩xd9₩x8d₩xa2₩xf5₩xb2<₩xba!₩x82w₩xda₩x1dy'
>>>
```

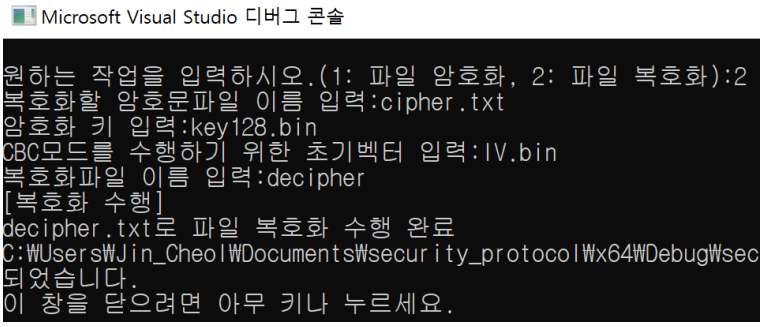
-Alice에게 받은 key128.bin을 복호화

key128.bin

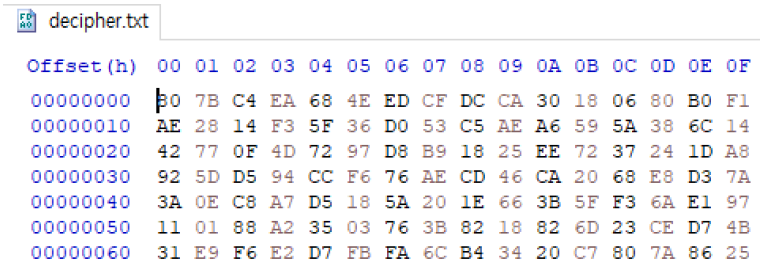
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	18	51	53	D9	8D	A2	F5	B2	3C	BA	6C	82	77	DA	1D	79

-RSA복호화를 통해 원래의 대칭키를 얻어냄

7)암호화된 파일 cipher.txt를 security_protocol로 AES CBC 복호화

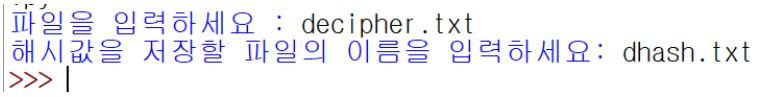


-cipher.txt를 key128.bin을 키값으로, IV.bin을 초기벡터로 설정하여 복호화 수행

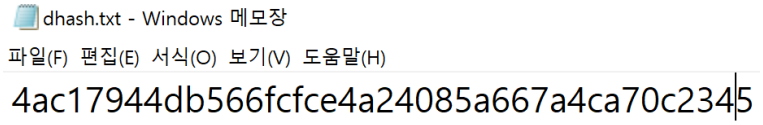


-복호화된 파일은 decipher.txt로 저장됨

8)sha1을 이용하여 복호화된 파일이 Alice가 보내려는 평문이 맞는지 검사



복호화된 파일인 decipher.txt의 해시값을 dhash.txt에 저장



-해시값이 Alice가 보내준 해시값과 동일하므로 파일이 손상되지 않았다는 것을 알 수 있음