REPORT

암호학 프로젝트5 RSA-OAEP/PSS

과목명	암호학
담당교수	오희국 교수님
학생이름	지현도
학과	컴퓨터학부
학번	2021004866
제출일	2023/11/20

HANYANG UNIVERSITY

*본인이작성한함수에대한설명:

1. static const size_t SHA2SIZE[6]

```
/*
 * SHA-2 function index list
 */
#define SHA224      0
#define SHA256      1
#define SHA384      2
#define SHA512      3
#define SHA512_224  4
#define SHA512_256  5

static const size_t SHA2SIZE[6]={
    28,32,48,64,28,32
};

```

어떤 SHA를 사용하는지에 따라 해당되는 인덱스 번호를 활용하여 해시의 크기를 정해주는 함수를 pkcs.h에 추가하였습니다.

이를 통해 SHA2SIZE[sha2_ndx]로 쉽게 크기를 가져올 수 있습니다. 기본크기는 비트를 8로나눈 바이트로 설정하였습니다.

2. void sha(const void *data, size_t len, unsigned char *digest, int sha2_ndx)

```
19 void sha(const void *data, size_t len, unsigned char *digest, int sha2_ndx)
20 {
21     switch(sha2_ndx){
22         case SHA224:
23             sha224(data,len,digest);
24             break;
25         case SHA256:
26             sha256(data,len,digest);
27             break;
28         case SHA384:
29             sha384(data,len,digest);
30             break;
31         case SHA512:
32             sha512(data,len,digest);
33             break;
34         case SHA512_224:
35             sha512_224(data,len,digest);
36             break;
37         case SHA512_256:
38             sha512_256(data,len,digest);
39             break;
40     }
```

프로젝트에서 기본적으로 해시함수로 sha를 사용하므로

인자로받는 sha2_ndx 즉 사용될 sha종류에 따라 적절한 sha 시리즈를 사용하도록 함수를 설정해주었습니다.

3. static unsigned char *mgf(const unsigned char *seed, size_t seedLen, unsinged char *mask, size_t maskLen, int sha2_ndx)

```c
static unsigned char *mgf(const unsigned char *seed, size_t seedLen, unsigned char *mask, size_t maskLen, int sha2_ndx)
{
    uint32_t i, count, c;
    size_t hLen = SHA2SIZE[sha2_ndx];
    unsigned char *mgfIn, *msg;

    /*
     * maskLen 이 2^32*hLen보다큰지 확인
     */
    if (maskLen > 0x0100000000 * hLen)
        return NULL;

    if ((mgfIn = (unsigned char *)malloc(seedLen + 4)) == NULL)
        return NULL;
    memcpy(mgfIn, seed, seedLen);
    count = maskLen / hLen + (maskLen % hLen ? 1 : 0);
    if ((msg = (unsigned char *)malloc(count * hLen)) == NULL){
        free(mgfIn);
        return NULL;
    }
    for (i = 0; i < count; i++) {
        c = i;
        mgfIn[seedLen + 3] = c & 0x000000ff; c >>= 8;
        mgfIn[seedLen + 2] = c & 0x000000ff; c >>= 8;
        mgfIn[seedLen + 1] = c & 0x000000ff; c >>= 8;
        mgfIn[seedLen] = c & 0x000000ff;
        (*sha)(mgfIn, seedLen + 4, msg + i * hLen,sha2_ndx);
    }

    memcpy(mask, msg, maskLen);
    free(mgfIn);
    free(msg);
    return mask;
}
```

RSA-OAEP와 RSA-PSS에서 사용하기 위해 필요한 mgf함수입니다.

4. rsaes_oaep_encrypt(const void *m, size_t mLen, const void *label, const void *e, const void *n, const void *c, int sha2_ndx)

```c
int rsaes_oaep_encrypt(const void *m, size_t mLen, const void *label, const void *e, const void *n, void *c, int sha2_ndx) {

    if (strlen(label) >= 0x1fffffffffffffff)
        return PKCS_LABEL_TOO_LONG;
    // 라벨 길이 제한 초과(2^64비트 즉, 2^61바이트보다 크면 안됨)

    size_t hLen;
    unsigned char *lHash;

    hLen = SHA2SIZE[sha2_ndx];
    lHash = malloc(sizeof(unsigned char) * hLen);

    sha(label, strlen(label), lHash, sha2_ndx);


    if (mLen > RSAKEYSIZE / 8 - 2 * hLen - 2)
        return PKCS_MSG_TOO_LONG;
    // 메세지가 너무 길면 오류메세지 출력

    // psLen 즉 PaddingString을 생성
    size_t psLen = RSAKEYSIZE / 8 - 2 - 2 * hLen - mLen;

    unsigned char *PaddingString = calloc(psLen, sizeof(unsigned char));


    // 주어진 변수들 + psLen으로 DB(DataBlock)생성
    size_t dbLen = hLen + psLen + 1 + mLen;
    unsigned char *DataBlock = malloc(sizeof(unsigned char) * dbLen);

    // memcpy를 활용하여 DataBlock에 순서대로 lHash, PaddingStirng, 01(0x01), Message를 붙여준다
    unsigned char temp[1] = {0x01};
    memcpy(DataBlock, lHash, hLen);
    memcpy(DataBlock + hLen, PaddingString, psLen);
    memcpy(DataBlock + hLen + psLen, temp, 1);
    memcpy(DataBlock + hLen + psLen + 1, m, mLen);


    // 난수 byte 문자열 seed를 생성. 이때 arc4random_buf를 사용 (openssl사용시 RAND_BYTE사용가능)
    unsigned char *seed = malloc(sizeof(unsigned char) * hLen);
    arc4random_buf(seed, hLen);

    // seed를 MGF에 통과시켜 dbMask를 생성
    unsigned char *dbMask = malloc(sizeof(unsigned char) * dbLen);
    mgf( seed, hLen,dbMask, dbLen, sha2_ndx);

    // mgf(dbMask) +(XOR) DataBlock으로 MaskedDataBlock을 생성
    unsigned char *MaskedDataBlock = malloc(sizeof(unsigned char) * dbLen);
    for (int i = 0; i < dbLen; i++) {
        MaskedDataBlock[i] = dbMask[i] ^ DataBlock[i];
    }

    // MaskedDataBlock을 MGF에 통과하여 seedMask를 생성
    unsigned char *seedMask = malloc(sizeof(unsigned char) * hLen);
    mgf(MaskedDataBlock, dbLen, seedMask, hLen, sha2_ndx);

    // mgf(seedMask) +(XOR) seed로 MaskedSeed을 생성
    unsigned char *MaskedSeed = malloc(sizeof(unsigned char) * hLen);
    for (int i = 0; i < hLen; i++) {
        MaskedSeed[i] = seedMask[i] ^ seed[i];
    };

    // Encoded Message에 차례대로 00(0x00), MaskedSeed, MaskedDataBlock을 붙여준다
    unsigned char *EncodedMessage = malloc(sizeof(unsigned char) * RSAKEYSIZE / 8);
    temp[0] = 0x00;
    memcpy(EncodedMessage, temp, 1);
    memcpy(EncodedMessage + 1, MaskedSeed, hLen);
    memcpy(EncodedMessage + 1 + hLen, MaskedDataBlock, dbLen);
```

```
    // EM를 rsa로 암호화
    int rsa_result = rsa_cipher(EncodedMessage, e, n);
    if(rsa_result != 0)
        return rsa_result;

    // 암호화된 EM을 c에 저장
    memcpy(c, EncodedMessage, (RSAKEYSIZE / 8));

    // 메모리 할당 해제
    free(lHash);
    free(PaddingString);
    free(DataBlock);
    free(MaskedDataBlock);
    free(seed);
    free(MaskedSeed);
    free(dbMask);
    free(seedMask);
    free(EncodedMessage);
    return 0;
}
```

길이가 mLen 인 메세지를 공개키(e,n)으로 암호화한 결과를 c에 저장해준다. 과정 중에 오류가 발생하면 오류 종류에 따라 오류코드를 반환해준다.

난수는 arc4random_buf를 사용하여 생성해주었고, 메모리 처리를 위해 동적할당과 memcpy를 사용해주었습니다.

5. int rsaes_oaep_decrypt(void *m, size_t *mLen, const void *label, const void *d, const void *n, const void *c, int sha2_ndx){

```
int rsaes_oaep_decrypt(void *m, size_t *mLen, const void *label, const void *d, const void *n, const void *c, int sha2_ndx) {

    if(strlen(label) >= 0x1fffffffffffffff)
        return PKCS_LABEL_TOO_LONG;
    // 라벨 길이 제한 초과

    //RSA 복호화
    unsigned char *encodedMessage = malloc(sizeof(unsigned char) * (RSAKEYSIZE/8));
    memcpy(encodedMessage, c, sizeof(unsigned char) * (RSAKEYSIZE/8));

    int rsa_result = rsa_cipher(encodedMessage, d, n);
    if(rsa_result != 0)
        return rsa_result;

    if(encodedMessage[0] != 0x00)
        return PKCS_INITIAL_NONZERO;
    // Encoded Message의 첫번째 바이트가 0이 아님

    // 복호화 과정 - 기존 seed, dataBlock 복원
    unsigned char *maskedSeed = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
    memcpy(maskedSeed, encodedMessage + 1, sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);

    unsigned char *maskedDataBlock = malloc(sizeof(unsigned char) * (RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1));
    memcpy(maskedDataBlock, encodedMessage + SHA2SIZE[sha2_ndx] + 1, sizeof(unsigned char) * (RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1));

    unsigned char *seed = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
    unsigned char *dataBlock = malloc(sizeof(unsigned char) * (RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1));
    mgf( maskedDataBlock, RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1,seed, SHA2SIZE[sha2_ndx], sha2_ndx);

    for (int i = 0; i < SHA2SIZE[sha2_ndx]; ++i)
        seed[i] ^= maskedSeed[i];

    mgf(seed, SHA2SIZE[sha2_ndx],dataBlock, RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1, sha2_ndx);

    for (int i = 0; i < RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1; ++i)
        dataBlock[i] ^= maskedDataBlock[i];


     // 원래의 메세지 복원
     unsigned char *labelHash = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
     memcpy(labelHash, dataBlock, sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);

     unsigned char *labelHash_In = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
     sha(label, strlen(label), labelHash_In,sha2_ndx);

     if(memcmp(labelHash, labelHash_In, SHA2SIZE[sha2_ndx]) != 0)
         return PKCS_HASH_MISMATCH; // label hash가 다름

     // padingString 확인(0x01이 맞는지)
     size_t ptr = SHA2SIZE[sha2_ndx];
     for(;ptr < RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1 && dataBlock[ptr] == 0x00; ++ptr);
     unsigned char divider = ptr < RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1 ? dataBlock[ptr] : 0x00;

     if(divider != 0x01)
         return PKCS_INVALID_PS;
     // paddingString 뒤에 붙는 값이 0x01이 아님

     // 최종 메세지 복호화
     *mLen = RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1 - ++ptr;
     memcpy(m, dataBlock + ptr, sizeof(char) * *mLen);

     // 메모리 할당 해제
     free(dataBlock);
     free(encodedMessage);
     free(maskedDataBlock);
     free(seed);
     free(maskedSeed);
     free(labelHash);
     free(labelHash_In);
     return 0;
}
```

암호문 c를 개인키(d,n)을 사용하여 원본 메세지 m과 길이 len을 회복한다. Label과 sha2_ndx는 암호화할때 사용한 것과 일치한다.

메모리 처리를 위해 동적할당과 memcpy를 사용하였습니다.

6. int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, const void *s, int sha2_ndx)

```
int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void *s, int sha2_ndx)
{
    if(mLen > 0x1fffffffffffffff)
        return PKCS_MSG_TOO_LONG;
    // mLen길이 제한 초과(2^64비트 즉, 2^61바이트보다 크면 안됨)

    // mHash 생성
    unsigned char mHash[SHA2SIZE[sha2_ndx]];
    sha(m, mLen, mHash,sha2_ndx);

    // arc4random_buf로 난수의 salt 생성
    unsigned salt[SHA2SIZE[sha2_ndx]]; // salt의 길이는 해시 길이와 같음
    arc4random_buf(salt, SHA2SIZE[sha2_ndx]);

    // 0x00(00) 8바이트와 mHash, salt를 이어붙여 mPrime(m')생성
    unsigned char mPrime[8+2*SHA2SIZE[sha2_ndx]];
    memset(mPrime, 0x00, 8);
    memcpy(mPrime+8, mHash, SHA2SIZE[sha2_ndx]);
    memcpy(mPrime+8+SHA2SIZE[sha2_ndx], salt, SHA2SIZE[sha2_ndx]);

    // mPrime을 해시를 통해 H생성
    unsigned char H[SHA2SIZE[sha2_ndx]];
    sha(mPrime, 8+2*SHA2SIZE[sha2_ndx], H,sha2_ndx);

    //DB 생성
    int DB_SIZE = RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1;
    unsigned char DB[DB_SIZE];

    memset(DB, 0, DB_SIZE - SHA2SIZE[sha2_ndx] -1); // ps
    DB[DB_SIZE-SHA2SIZE[sha2_ndx]-1] = 0x01; // 0x01
    memcpy(DB + DB_SIZE-SHA2SIZE[sha2_ndx], salt, SHA2SIZE[sha2_ndx]); // salt

    // H를 MGF에 통과시켜 H마스크 생성
    unsigned char MaskedH[DB_SIZE];
    mgf(H, SHA2SIZE[sha2_ndx], MaskedH, DB_SIZE, sha2_ndx);
```

```
385     // MaskedH와 DB XOR 연산하여 maskedDB생성
386     unsigned char maskedDB[DB_SIZE];
387     for(int i=0; i<DB_SIZE; i++){
388         maskedDB[i] = DB[i] ^ MaskedH[i];
389     }
390
391     if(DB_SIZE + SHA2SIZE[sha2_ndx] + 1 > RSAKEYSIZE/8)
392         return PKCS_HASH_TOO_LONG;
393     // H가 EM의 길이보다 크면 수용불가능
394
395     //EM 생성
396     unsigned char EM[RSAKEYSIZE/8];
397     memcpy(EM, maskedDB, DB_SIZE);
398     memcpy(EM+DB_SIZE, H, SHA2SIZE[sha2_ndx]);
399     EM[RSAKEYSIZE/8-1] = 0xbc;
400
401     // EM의 첫 비트가 1이면 0으로 바꿔줌
402     if((EM[0]>>7) & 1) EM[0] = 0x00;
403
404     // 키 사용하여 암호화
405     if(rsa_cipher(EM, d, n) == PKCS_MSG_OUT_OF_RANGE)
406         return PKCS_MSG_OUT_OF_RANGE;
407     memcpy(s, EM, RSAKEYSIZE/8);
408
409     return 0;
410 }
```

길이가 mLen 인 메세지를 개인키(d,n)으로 서명한 후 결과를 s에 저장한다. 과정 중에 오류가 발생하면 오류 종류에 따라 오류코드를 반환해준다.

난수는 arc4random_buf를 사용하여 생성해주었고, 메모리 처리를 위해 동적할당과 memcpy를 사용해주었습니다.

7. int rsassa_pss_verify(const void *m, size_t mLen, const void *e, const void *n, const void *s, int sha2_ndx)

```
int rsassa_pss_verify(const void *m, size_t mLen, const void *e, const void *n, const void *s, int sha2_ndx)
{
    unsigned char EM[RSAKEYSIZE/8];
    memcpy(EM, s, RSAKEYSIZE/8);

    // 키 사용하여 복호화
    if(rsa_cipher(EM, e, n) == PKCS_MSG_OUT_OF_RANGE)
        return PKCS_MSG_OUT_OF_RANGE;

    // 오류 검증
    if(EM[RSAKEYSIZE/8-1] ^ 0xbc) return PKCS_INVALID_LAST;
    if((EM[0] >> 7) & 1) return PKCS_INVALID_INIT;

    // maskedDB 추출
    int DB_SIZE = RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1;
    unsigned char maskedDB[DB_SIZE];
    memcpy(maskedDB, EM, DB_SIZE);

    // H 추출
    unsigned char H[SHA2SIZE[sha2_ndx]];
    memcpy(H, EM+DB_SIZE, SHA2SIZE[sha2_ndx]);

    // MaskedH 복원
    unsigned char MaskedH[DB_SIZE];
    mgf(H, SHA2SIZE[sha2_ndx], MaskedH, DB_SIZE, sha2_ndx);

    // DB 복원
    unsigned char DB[DB_SIZE];
    DB[0] = 0x00;
    for(int i=1; i<DB_SIZE; i++){
        DB[i] = maskedDB[i] ^ MaskedH[i];
    }

    // salt 복원
    unsigned char salt[SHA2SIZE[sha2_ndx]];
    memcpy(salt, DB+DB_SIZE-SHA2SIZE[sha2_ndx], SHA2SIZE[sha2_ndx]);


    // DB 앞 부분이 0x0000..00||0x01과 일치하는지 확인
    if(DB[DB_SIZE-SHA2SIZE[sha2_ndx]-1] ^ 0x01) return  PKCS_INVALID_PD2 ;
    for(int i=0; i<DB_SIZE - SHA2SIZE[sha2_ndx] - 1; i++){
        if(DB[i] ^ 0x00) return PKCS_INVALID_PD2;
    }

    // 주어진 m으로 mHash 생성
    unsigned char mHash[SHA2SIZE[sha2_ndx]];
    sha(m, mLen, mHash,sha2_ndx);

    // mPrime 생성
    unsigned char mPrime[8+2*SHA2SIZE[sha2_ndx]];
    memset(mPrime, 0x00, 8);
    memcpy(mPrime+8, mHash, SHA2SIZE[sha2_ndx]);
    memcpy(mPrime+8+SHA2SIZE[sha2_ndx], salt, SHA2SIZE[sha2_ndx]);

    // mPrime Hash 생성
    unsigned char mPrimeHash[SHA2SIZE[sha2_ndx]];
    sha(mPrime, 8+2*SHA2SIZE[sha2_ndx], mPrimeHash,sha2_ndx);

    // mPrime Hash와 H 비교
    if(memcmp(mPrimeHash, H, SHA2SIZE[sha2_ndx]) != 0) return PKCS_HASH_MISMATCH;

    return 0;
}
```

길이가 mLen 인 메세지에 대한 서명s가 맞는지 주어진 공개

키(e,n)으로 검증한다. 검증에 성공하지못한다면 그에 맞는오류코드를 반환해준다.

위 함수에서도 또한 메모리 처리를 위해 memset, memcpy, memcmp를 사용해주었습니다.

*컴파일 과정

```
jihyeondo@192 proj#5-1 % make
gcc -Wall -O3 -c pkcs.c
gcc -o test test.o pkcs.o sha2.o -lgmp
jihyeondo@192 proj#5-1 %
```

오류없이 정상적으로 컴파일되는 모습이다.

*실행 결과물

```
jihyeondo@192 proj#5-1 % ./test
e = 40a50ff70fbd9d2201a1e5b9f2736648bf9286262b84b504feb1c4b077f39a578b44665f21d21bb00ab90a63e3de346170e53852b53acb0659922227c7fe371630b
1dfaebf6a6c0b2f20b4866ab39f9eb86f59e1e89024ef48ae9eb4346dbf4b43d9ed1a5369b531ea4f6a96dd062e4fa1b1329f8ce51ab69938422a19311076dcc900f149
6cb43b31b82eb6d6d8df26d2bd8f71baf724266de397dca5586557b6a0abcd16d22649ffb2992a7b2fbd7ecb809b623ff5295c155fe16d1048a2edeed094ff9aafc28cc
5d641a5644b48338ae7723e3f974898e6cb0c2b60ad967d9ce2efa27b565446f1be88df5b53c8b82f3272db2a61011e5ccee68ae783b2b7
d = 2e34546cf61565627bb5de34f8e95bb187d1970e2015684a7075b2b24f78c25ee16ad67e30d2d1d8a82ed7d7484a10f90c25d0cc93c360c3d6e508341dabf528a3d
37366232a975784f095f8674258ae64c3013f9e6eedaf7bb4754f9c0eca5bb6516f45eca6358794cc74079dffa8967ed7a4526469cf97c3979fdd0939f2f14e71c8fbdd
78acc3234027db441fc09df159e02e0eed80d769cbaa4d6026c2e94d2950be2048c312c10a1dc4f38ea710fcfc03719d0d9752e53c80eef2d4478f9be0426dc73e8ab09
d6beda33f02a5f156dbdc402eee28e23d7340b70fc138841b40f76bb9d3053d0d2b0cabb6076deb3283ff46216c9d7e5970f86d0a3872eb
n = 857d923ea865ce4f7c91b050a253ef7e86abf1d91b3bd53b7d93ded592cea11a81db7f6fc3325d5f1a631bbf93f9a5709fec7f3a93d218cfbabc17aa3d76e4f2e2b
1a19ed7e78265bca16c58ce223128261bce38e85ccc83d5e2bc4f79144e618ca8e1edf3f842ca581c9008a7ab140c4b0b584f78d76485b7a1aafc2161f3adf4662035f7
2cec565ea5cb33f97801272ff0a2977e41ec9b4e634d1d194982a77b95a6488be05b23fee9926a3e26a39bb5310358583e20d4e19d031d23165539c600bd36bd2ee16d2
1d0505c4b8fc0d088b2054ea10d5b205260b469d82568252ce5290f5a347941ed6e4057626562b20da3b9f4de00f87dd2978a5f0f799597
---
c = 082cba0eef163fc8383a39c63fdefc877e87211fcf53b395ee94e43efeb7380edcd77dc74b6b5ab21046c746df3a1d7d068bf37a7b4e0efff7be06169285dd4f0f7
8c9cc743ef48fa9f8872cc341cfa613e82f458cbfd36bcd4dc0b1f576f4d9b3465dd4711f3cd16c4dbbe05030c9c0731e5acfc991adcb100331f62ad7534d8929e8d5d2
353ce124bd8a400f1236ee7794ad6957aaf8bc45847a0f63dd305d711a1c1e84a6962e891ca2b7016c461b71a9e513a86749495d7fefea4b7be57d51cbea6eac1318265
ebc7f18e3b372821168390a297bd2734fe759565eb8f4dad629a7273b0fd02d90dc50d6787c6a1da75ae420c31ae75a5b48aaa2b61d9780
m = 73616d706c65206461746100
msg = sample data, len = 12 -- PASSED
---
c = 6c2081c71cd86a235aa3ec5121c380276fe8c9689ac97d3a919bc773378a315383e9b274ecbb2dee1932d767342bd02f06b41d8b3281239c031626bd011a9973f8f
4a3f36c5832d76bc4c18b5edfd6773cb2615687bca93273484f3ece7e2cd1aa845ae2072d84ec18887157e58bb43c1d13de965ab9241e69cb90655a805503868a9854c2
41d55c1f372065b2c38ac392c672fd2281c6b641c31aa1a85249cca95a10bb8d7701af3873e5fb65bd8a1268f8a9cc45672173eeb14117b53528559e492cc780e6266d4
ce15844e8355e86602b3e8210060844b4958b1c3da447f789de0ff9130ef790a5e88fa1f3f5e28a721507dec48d601104c2b915444b7481
m = 6d617820646174610 06c6162656c006d61782b20646174610045 6e6372797074696f6e204572726f723a2025642c206d65737361676520697320746f6f206c6f6e6
7202d2d20504153534544 0a2d2d2d0a0044656372797074696f6e204572726f723a206d657373676165206c656e6774682025 7a7520697320 6e6f74207a65726f202d2d
204641494c45440a006b6f7265616e20706f6574000a6d7367203d2025732c206c656e203d20257a75202d2d20504153534544 0a0045696e73
msg = max data, len = 190 -- PASSED
---
Encryption Error: 2, message is too long -- PASSED
---
c = 78f1eb027b352cf697ceaa092ef512088bc7d59f117d4f5acb00405700a0ca2d737d4fa3c319c970d80ac41b8c42482baea46ebbd92f925dc0e120fc05e030eb2c9
e5210df9099631af3e9e7a5426c091ce56eb150fb5a81da76087c1ac8a27a7060d43fa2d4e88b3adbced1e0d63ea302d77c68fa3ac1e812a7041678d0abfa4556adf82e
43772cf1932dab4a1227fc8f5791721809ac0a8596289a78992679 6a30798e0a4b1fe6cdc9d76bce5dd98610434ef6fdd7263982e4f364abceeed1dd6a22a5c35c79fb1
6731870779317456648e5b436622fa4d18910ac531dbba7b911b877f5d04d21b439b691e163f7159da80b22df053b0abdcb873a17e0021a
Empty message -- PASSED
---
m = ec9ca4eb8f99eca3bc
msg = 윤동주, len = 9 -- PASSED
msg = 배움은 경험에서 얻는다. 경험하지 않은 것은 정보에 불과하다. – 아인슈타인 -- PASSED
---
```

테스트코드의 모든 테스트를 오류없이 통과한 모습입니다.

RSA-OAEP, RSA-PSS 테스트 결과가 예상출력과는 먼저 출력되는 에러메세지가 다르긴하지만 모두 정상적으로 PASSED된 걸로 보아 정상적으로 판별되는 것 같습니다.

알고리즘의 효율 문제인지 CPU 실행시간은 교수님의 샘플보다 다소 느린 것 같습니다.

*프로젝트 소감 및 부족한 점

수업에서 배운 OAEP와 PSS를 실제로 구현해보았는데, 메모리 할당과 효율적인 처리에 아직 미숙한 점이 많은 것 같습니다. 더 가독성 높고 효율적인 코드처리를 고민해볼 필요가 있을 것 같습니다.

또한 해당 프로젝트에서는 SHA2계열을 사용하였는데 안전성을 위해 SHA3을 활용하여 구현하면 어떨까 생각해보았는데, 호환성 측면도 걱정되고 항상 SHA3가 성능에서 우수한 것은 아닐 수도 있다는 생각이 들어서 이부분은 더 학습이 필요할 것 같습니다.