

REPORT

암호학 프로젝트5 RSA-OAEP/PSS



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

*본인이작성한함수에대한설명:

1. static const size_t SHA2SIZE[6]

```
12  /*
13   * SHA-2 function index list
14   */
15  #define SHA224      0
16  #define SHA256      1
17  #define SHA384      2
18  #define SHA512      3
19  #define SHA512_224  4
20  #define SHA512_256  5
21
22  static const size_t SHA2SIZE[6]={
23      28,32,48,64,28,32
24  };
25
```

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, unsigned char *mask, size_t maskLen, int sha2_ndx)

```
150 static unsigned char *mgf(const unsigned char *seed, size_t seedLen, unsigned char *mask, size_t maskLen, int sha2_ndx)
151 {
152     uint32_t i, count, c;
153     size_t hLen = SHA2SIZE[sha2_ndx];
154     unsigned char *mgfIn, *msg;
155
156     /*
157      * maskLen 이 2^32*hLen보다큰지 확인
158      */
159     if (maskLen > 0x0100000000 * hLen)
160         return NULL;
161
162     if ((mgfIn = (unsigned char *)malloc(seedLen + 4)) == NULL)
163         return NULL;
164     memcpy(mgfIn, seed, seedLen);
165     count = maskLen / hLen + (maskLen % hLen ? 1 : 0);
166     if ((msg = (unsigned char *)malloc(count * hLen)) == NULL){
167         free(mgfIn);
168         return NULL;
169     }
170     for (i = 0; i < count; i++) {
171         c = i;
172         mgfIn[seedLen + 3] = c & 0x000000ff; c >>= 8;
173         mgfIn[seedLen + 2] = c & 0x000000ff; c >>= 8;
174         mgfIn[seedLen + 1] = c & 0x000000ff; c >>= 8;
175         mgfIn[seedLen] = c & 0x000000ff;
176         (*sha)(mgfIn, seedLen + 4, msg + i * hLen, sha2_ndx);
177     }
178
179     memcpy(mask, msg, maskLen);
180     free(mgfIn);
181     free(msg);
182     return mask;
183 }
```

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)

```
185 int rsaes_oaep_encrypt(const void *m, size_t mLen, const void *label, const void *e, const void *n, void *c, int sha2_ndx) {
186
187     if (strlen(label) >= 0xffffffffffff)
188         return PKCS_LABEL_TOO_LONG;
189     // 라벨 길이 제한 초과(2^64비트 즉, 2^61바이트보다 크면 안됨)
190
191     size_t hLen;
192     unsigned char *lHash;
193
194     hLen = SHA2SIZE[sha2_ndx];
195     lHash = malloc(sizeof(unsigned char) * hLen);
196
197     sha(label, strlen(label), lHash, sha2_ndx);
198
199
200     if (mLen > RSAKEYSIZE / 8 - 2 * hLen - 2)
201         return PKCS_MSG_TOO_LONG;
202     // 메시지가 너무 길면 오류메세지 출력
203
204     // psLen 즉 PaddingString을 생성
205     size_t psLen = RSAKEYSIZE / 8 - 2 - 2 * hLen - mLen;
206
207     unsigned char *PaddingString = calloc(psLen, sizeof(unsigned char));
208
209
210     // 주어진 변수들 + psLen으로 DB(DataBlock)생성
211     size_t dbLen = hLen + psLen + 1 + mLen;
212     unsigned char *DataBlock = malloc(sizeof(unsigned char) * dbLen);
213
214     // memcpy를 활용하여 DataBlock에 순서대로 lHash, PaddingString, 01(0x01), Message를 붙여준다
215     unsigned char temp[1] = {0x01};
216     memcpy(DataBlock, lHash, hLen);
217     memcpy(DataBlock + hLen, PaddingString, psLen);
218     memcpy(DataBlock + hLen + psLen, temp, 1);
219     memcpy(DataBlock + hLen + psLen + 1, m, mLen);
220
221
222     // 난수 byte 문자열 seed를 생성. 이때 arc4random_buf를 사용 (openssl사용시 RAND_BYTE사용가능)
223     unsigned char *seed = malloc(sizeof(unsigned char) * hLen);
224     arc4random_buf(seed, hLen);
225
226     // seed를 MGF에 통과시켜 dbMask를 생성
227     unsigned char *dbMask = malloc(sizeof(unsigned char) * dbLen);
228     mgf(seed, hLen, dbMask, dbLen, sha2_ndx);
229
230     // mgf(dbMask) +(XOR) DataBlock으로 MaskedDataBlock을 생성
231     unsigned char *MaskedDataBlock = malloc(sizeof(unsigned char) * dbLen);
232     for (int i = 0; i < dbLen; i++) {
233         MaskedDataBlock[i] = dbMask[i] ^ DataBlock[i];
234     }
235
236     // MaskedDataBlock을 MGF에 통과하여 seedMask를 생성
237     unsigned char *seedMask = malloc(sizeof(unsigned char) * hLen);
238     mgf(MaskedDataBlock, dbLen, seedMask, hLen, sha2_ndx);
239
240     // mgf(seedMask) +(XOR) seed로 MaskedSeed을 생성
241     unsigned char *MaskedSeed = malloc(sizeof(unsigned char) * hLen);
242     for (int i = 0; i < hLen; i++) {
243         MaskedSeed[i] = seedMask[i] ^ seed[i];
244     };
245
246     // Encoded Message에 차례대로 00(0x00), MaskedSeed, MaskedDataBlock을 붙여준다
247     unsigned char *EncodedMessage = malloc(sizeof(unsigned char) * RSAKEYSIZE / 8);
248     temp[0] = 0x00;
249     memcpy(EncodedMessage, temp, 1);
250     memcpy(EncodedMessage + 1, MaskedSeed, hLen);
251     memcpy(EncodedMessage + 1 + hLen, MaskedDataBlock, dbLen);
```

```

252 // EM를 rsa로 암호화
253 int rsa_result = rsa_cipher(EncodedMessage, e, n);
254 if(rsa_result != 0)
255     return rsa_result;
256
257 // 암호화된 EM을 c에 저장
258 memcpy(c, EncodedMessage, (RSAKEYSIZE / 8));
259
260 // 메모리 할당 해제
261 free(lHash);
262 free(paddingString);
263 free(DataBlock);
264 free(MaskedDataBlock);
265 free(seed);
266 free(MaskedSeed);
267 free(dbMask);
268 free(seedMask);
269 free(EncodedMessage);
270 return 0;
271 }

```

길이가 mLen 인 메시지를 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){

```

273 int rsaes_oaep_decrypt(void *m, size_t *mLen, const void *label, const void *d, const void *n, const void *c, int sha2_ndx) {
274
275     if(strlen(label) >= 0xffffffffffffffff)
276         return PKCS_LABEL_TOO_LONG;
277     // 라벨 길이 제한 초과
278
279     //RSA 복호화
280     unsigned char *encodedMessage = malloc(sizeof(unsigned char) * (RSAKEYSIZE/8));
281     memcpy(encodedMessage, c, sizeof(unsigned char) * (RSAKEYSIZE/8));
282
283     int rsa_result = rsa_cipher(encodedMessage, d, n);
284     if(rsa_result != 0)
285         return rsa_result;
286
287     if(encodedMessage[0] != 0x00)
288         return PKCS_INITIAL_NONZERO;
289     // Encoded Message의 첫번째 바이트가 0이 아님
290
291     // 복호화 과정 - 기존 seed, dataBlock 복원
292     unsigned char *maskedSeed = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
293     memcpy(maskedSeed, encodedMessage + 1, sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
294
295     unsigned char *maskedDataBlock = malloc(sizeof(unsigned char) * (RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1));
296     memcpy(maskedDataBlock, encodedMessage + SHA2SIZE[sha2_ndx] + 1, sizeof(unsigned char) * (RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1));
297
298     unsigned char *seed = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
299     unsigned char *dataBlock = malloc(sizeof(unsigned char) * (RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1));
300     mgf(maskedDataBlock, RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1, seed, SHA2SIZE[sha2_ndx], sha2_ndx);
301
302     for (int i = 0; i < SHA2SIZE[sha2_ndx]; ++i)
303         seed[i] ^= maskedSeed[i];
304
305     mgf(seed, SHA2SIZE[sha2_ndx], dataBlock, RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1, sha2_ndx);
306
307     for (int i = 0; i < RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1; ++i)
308         dataBlock[i] ^= maskedDataBlock[i];
309
310     // 원래의 메세지 복원
311     unsigned char *labelHash = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
312     memcpy(labelHash, dataBlock, sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
313
314     unsigned char *labelHash_In = malloc(sizeof(unsigned char) * SHA2SIZE[sha2_ndx]);
315     sha(label, strlen(label), labelHash_In, sha2_ndx);
316
317     if(memcmp(labelHash, labelHash_In, SHA2SIZE[sha2_ndx]) != 0)
318         return PKCS_HASH_MISMATCH; // label hash가 다름
319
320     // paddingString 확인(0x01이 맞는지)
321     size_t ptr = SHA2SIZE[sha2_ndx];
322     for(; ptr < RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1 && dataBlock[ptr] == 0x00; ++ptr);
323     unsigned char divider = ptr < RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1 ? dataBlock[ptr] : 0x00;
324
325     if(divider != 0x01)
326         return PKCS_INVALID_PS;
327     // paddingString 뒤에 붙는 값이 0x01이 아님
328
329     // 최종 메세지 복호화
330     *mLen = RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1 - ++ptr;
331     memcpy(m, dataBlock + ptr, sizeof(char) * *mLen);
332
333     // 메모리 할당 해제
334     free(dataBlock);
335     free(encodedMessage);
336     free(maskedDataBlock);
337     free(seed);
338     free(maskedSeed);
339     free(labelHash);
340     free(labelHash_In);
341     return 0;
342 }

```

암호문 c 를 개인키(d, n)을 사용하여 원본 메시지 m 과 길이 len 을 회복한다. $Label$ 과 $sha2_ndx$ 는 암호화할때 사용한 것과 일치한다.

메모리 처리를 위해 동적할당과 `memcpy`를 사용하였습니다.

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

```
349 int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void *s, int sha2_ndx)
350 {
351     if(mLen > 0xffffffffffffffff)
352         return PKCS_MSG_TOO_LONG;
353     // mLen길이 제한 초과(2^64바이트 즉, 2^61바이트보다 크면 안됨)
354
355     // mHash 생성
356     unsigned char mHash[SHA2SIZE[sha2_ndx]];
357     sha(m, mLen, mHash, sha2_ndx);
358
359     // arc4random_buf로 난수의 salt 생성
360     unsigned salt[SHA2SIZE[sha2_ndx]]; // salt의 길이는 해시 길이와 같음
361     arc4random_buf(salt, SHA2SIZE[sha2_ndx]);
362
363     // 0x00(00) 8바이트와 mHash, salt를 이어붙여 mPrime(m')생성
364     unsigned char mPrime[8+2*SHA2SIZE[sha2_ndx]];
365     memset(mPrime, 0x00, 8);
366     memcpy(mPrime+8, mHash, SHA2SIZE[sha2_ndx]);
367     memcpy(mPrime+8+SHA2SIZE[sha2_ndx], salt, SHA2SIZE[sha2_ndx]);
368
369     // mPrime을 해시를 통해 H생성
370     unsigned char H[SHA2SIZE[sha2_ndx]];
371     sha(mPrime, 8+2*SHA2SIZE[sha2_ndx], H, sha2_ndx);
372
373     //DB 생성
374     int DB_SIZE = RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1;
375     unsigned char DB[DB_SIZE];
376
377     memset(DB, 0, DB_SIZE - SHA2SIZE[sha2_ndx] - 1); // ps
378     DB[DB_SIZE-SHA2SIZE[sha2_ndx]-1] = 0x01; // 0x01
379     memcpy(DB + DB_SIZE-SHA2SIZE[sha2_ndx], salt, SHA2SIZE[sha2_ndx]); // salt
380
381     // H를 MQF에 통과시켜 H마스크 생성
382     unsigned char MaskedH[DB_SIZE];
383     mgf(H, SHA2SIZE[sha2_ndx], MaskedH, DB_SIZE, sha2_ndx);
384     ...
```



```

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 메시지를 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)

```

416 int rsassa_pss_verify(const void *m, size_t mLen, const void *e, const void *n, const void *s, int sha2_ndx)
417 {
418     unsigned char EM[RSAKEYSIZE/8];
419     memcpy(EM, s, RSAKEYSIZE/8);
420
421     // 키 사용하여 복호화
422     if(rsa_cipher(EM, e, n) == PKCS_MSG_OUT_OF_RANGE)
423         return PKCS_MSG_OUT_OF_RANGE;
424
425     // 오류 검증
426     if(EM[RSAKEYSIZE/8-1] ^ 0xbc) return PKCS_INVALID_LAST;
427     if((EM[0] >> 7) & 1) return PKCS_INVALID_INIT;
428
429     // maskedDB 추출
430     int DB_SIZE = RSAKEYSIZE/8 - SHA2SIZE[sha2_ndx] - 1;
431     unsigned char maskedDB[DB_SIZE];
432     memcpy(maskedDB, EM, DB_SIZE);
433
434     // H 추출
435     unsigned char H[SHA2SIZE[sha2_ndx]];
436     memcpy(H, EM+DB_SIZE, SHA2SIZE[sha2_ndx]);
437
438     // MaskedH 복원
439     unsigned char MaskedH[DB_SIZE];
440     mgf(H, SHA2SIZE[sha2_ndx], MaskedH, DB_SIZE, sha2_ndx);
441
442     // DB 복원
443     unsigned char DB[DB_SIZE];
444     DB[0] = 0x00;
445     for(int i=1; i<DB_SIZE; i++){
446         DB[i] = maskedDB[i] ^ MaskedH[i];
447     }
448
449     // salt 복원
450     unsigned char salt[SHA2SIZE[sha2_ndx]];
451     memcpy(salt, DB+DB_SIZE-SHA2SIZE[sha2_ndx], SHA2SIZE[sha2_ndx]);
452
453     // DB 앞 부분이 0x0000..00||0x01과 일치하는지 확인
454     if(DB[DB_SIZE-SHA2SIZE[sha2_ndx]-1] ^ 0x01) return PKCS_INVALID_PD2 ;
455     for(int i=0; i<DB_SIZE - SHA2SIZE[sha2_ndx] - 1; i++){
456         if(DB[i] ^ 0x00) return PKCS_INVALID_PD2;
457     }
458
459     // 주어진 m으로 mHash 생성
460     unsigned char mHash[SHA2SIZE[sha2_ndx]];
461     sha(m, mLen, mHash, sha2_ndx);
462
463     // mPrime 생성
464     unsigned char mPrime[8+2*SHA2SIZE[sha2_ndx]];
465     memset(mPrime, 0x00, 8);
466     memcpy(mPrime+8, mHash, SHA2SIZE[sha2_ndx]);
467     memcpy(mPrime+8+SHA2SIZE[sha2_ndx], salt, SHA2SIZE[sha2_ndx]);
468
469     // mPrime Hash 생성
470     unsigned char mPrimeHash[SHA2SIZE[sha2_ndx]];
471     sha(mPrime, 8+2*SHA2SIZE[sha2_ndx], mPrimeHash, sha2_ndx);
472
473     // mPrime Hash와 H 비교
474     if(memcmp(mPrimeHash, H, SHA2SIZE[sha2_ndx]) != 0) return PKCS_HASH_MISMATCH;
475
476     return 0;
477 }
478

```

길이가 mLen 인 메시지에 대한 서명s가 맞는지 주어진 공개

키(e, n)으로 검증한다. 검증에 성공하지 못한다면 그에 맞는 오류코드를 반환해준다.

위 함수에서도 또한 메모리 처리를 위해 `memset`, `memcpy`, `memcmp`를 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 = 40a50fff70fbd9d2201a1e5b9ff2736648bf9286262b84b504feb1c4b077f39a578b44665f21d21bb00ab90a63e3de346170e53852b53acb0659922227c7fe371630b
1dfaebf6a6c0b2f20b4866ab39f9eb86f59e1e89024ef48ae9eb4346dbf4b43d9ed1a5369b531ea4f6a96dd062e4fa1b1329f8ce51ab69938422a19311076dcc900f149
6cb43b31b82eb6d6d8df26d2bd8f71baf724266de397dca5586557b6a0abcd16d22649fffb2992a7b2fbd7ecb809b623ff5295c155fe16d1048a2edeed094ff9aafc28cc
5d641a5644b48338ae7723e3f974898e6cb0c2b60ad967d9ce2efa27b565446f1be88df5b53c8b82f3272db2a61011e5ccee68ae783b2b7
d = 2e34546cf61565627bb5de34f8e95bb187d1970e2015684a7075b2b24f78c25ee16ad67e30d2d1d8a82ed7d7484a10f90c25d0cc93c360c3d6e508341dabf528a3d
37366232a975784f095f8674258ae64c3013f9e6eedaf7bb4754f9c0eca5bb6516f45eca6358794cc74079dffa8967ed7a4526469cf97c3979fdd0939f2f14e71c8fbdd
78acc3234027db441fc09df159e02e0eed80d769cbaa4d6026c2e94d2950be2048c312c10a1dc4f38ea710fcfc03719d0d9752e53c80eef2d4478f9be0426dc73e8ab09
d6beda33f02a5f156dbdc402eee28e23d7340b70fc138841b40f76bb9d3053d0d2b0cabb6076deb3283ff46216c9d7e5970f86d0a3872eb
n = 857d923ea865ce4f7c91b050a253ef7e86abf1d91b3bd53b7d93ded592cea11a81db7f6fc3325d5f1a631bbf93f9a5709fec7f3a93d218cfbabc17aa3d76e4f2e2b
1a19ed7e78265bca16c58ce223128261bce38e85ccc83d5e2bc4f79144e618ca8e1edf3f842ca581c9008a7ab140c4b0b584f78d76485b7a1aafc2161f3adf4662035f7
2cec565ea5cb33f97801272ff0a2977e41ec9b4e634d1d194982a77b95a6488be05b23fee9926a3e26a39bb5310358583e20d4e19d031d23165539c600bd36bd2ee16d2
1d0505c408fc0d088b2054ea10d5b205260b469d82568252ce5290f5a347941ed6e4057626562b20da3b9f4de00f87dd2978a5f0f799597
---
c = 082cba0eef163fc8383a39c63dfefc877e87211fcf53b395ee94e43efeb7380edcd77cd74b6b5ab21046c746df3a1d7d068bf37a7b4e0effff7be06169285dd4f0f7
8c9cc743ef48fa9f8872cc341cfa613e82f458cbfd36bcd4dc0b1f576f4d9b3465dd4711f3cd16c4dbbe05030c9c0731e5acfc991adcb100331f62ad7534d8929e8d5d2
353ce124bd8a400f1236ee7794ad6957aaf8bc45847a0f63dd305d711a1cle84a6962e891ca2b7016c461b71a9e513a86749495d7fefeab7be57d51cbeaeac1318265
ebc7f18e3b372821168390a297bd2734fe759565eb8f4dad629a7273b0fd02d90dc50d6787c6a1da75ae420c31ae75a5b48aaa2b61d9780
m = 73616d706c65206461746100
msg = sample data, len = 12 -- PASSED
---
c = 6c2081c71cd86a235aa3ec5121c380276fe8c9689ac97d3a919bc773378a315383e9b274ecbb2dee1932d767342bd02f06b41d8b3281239c031626bd011a9973f8f
4a3f36c5832d76bc4c18b5edfd6773cb2615687bca93273484f3ece7e2cd1aa845ae2072d84ec18887157e58bb43c1d13de965ab9241e69cb90655a805503868a9854c2
41d55c1f372065b2c38ac392c672fd2281c6b641c31aa1a85249cca95a10bb8d7701af3873e5fb65bd8a1268f8a9cc45672173eeb14117b53528559e492cc780e6266d4
ce15844e8355e86602b3e8210060844b4958b1c3da447f789de0ff9130ef790a5e88fa1f3f5e28a721507dec48d601104c2b915444b7481
m = 6d61782064617461006c6162656c006d61782b206461746100456e6372797074696f6e204572726f723a206d65737676165206c656e67746820257a75206973206e6f74207a65726f202d2d
7202d2d205041535345440a2d2d2d0a0044656372797074696f6e204572726f723a206d65737676165206c656e67746820257a75206973206e6f74207a65726f202d2d
2046441494c45440a006b6f7265616e20706f6574000a6d7367203d2025732c206c656e203d20257a75202d2d205041535345440a0045696e73
msg = max data, len = 190 -- PASSED
---
Encryption Error: 2, message is too long -- PASSED
---
c = 78f1eb027b352cf697ceaa092ef512088bc7d59f117d4f5acb00405700a0ca2d737d4fa3c319c970d80ac41b8c42482baea46ebbd92f925dc0e120fc05e030eb2c9
e5210df9099631af3e9e7a5426c091ce56eb150fb5a81da76087c1ac8a27a7060d43fa2d4e88b3adbced1e0d63ea302d77c68fa3ac1e812a7041678d0abfa4556adf82e
43772cf1932dab4a1227fc8f5791721809ac0a8596289a789926796a30798e0a4b1fe6cdc9d76bce5dd98610434ef6fdd7263982e4f364abceeed1dd6a2a25c35c79fb1
6731870779317456648e5b436622fa4d18910ac531dbba7b911b877f5d0421b439b691e163f7159da80b22df053b0abdc873a17e0021a
Empty message -- PASSED
---
m = ec9ca4eb8f99eca3bc
msg = 윤동주, len = 9 -- PASSED
msg = 벚꽃은 경험에서 얻는다. 경험하지 않은 것은 정보에 불과하다. - 아인슈타인 -- PASSED
---
```

```

RSAES-OAEP Random Testing.....No error found
! -- PASSED
---
s = 1e59233881c17302285e44eb90b674838a0337816d0f326d2de24adb595ef187e5f2d285db867a2941ccc3d25d26d8cb8a5878a753b6a7f5b3f41d30fff5992310fd
0808b03637690d3773bea79a603fe20c1ce1e9d1154f7d27476da62e99f12d46b24a58b3f696027f9ea72588df70942c49e0fc304c673e84539c42fd82301128ca37fb5
7ddf4e8c407747d45dbdf415afc4f2f4e0daae910197bae0377ee1f421e1183e0fcf1c43a6c0f9d501dba0aee3e84702d265708939a7df28eb45d6f23c023b303df1caa
b3bcc6e63dc5babdfb5ff5d45d37b15318cc3efce8f3517c567a91441bab1537596cfb575f8fdb75efa821645cf5da3d30b2e7c375b6a66
Valid Signature! -- PASSED
---
Hash Input Error: 10 -- PASSED
---
s = 19791534d53858d3e7866bf9381507683b73c26da0502929e19a5acd66b59ba26b45a9db660ceac87db7b90fcf3b84e5219ed16322faf9e5e13711265c7cb4ae74
868b87b90f40e448de1594dd1eba47712d35c1c7fd34c9aba0635c40199fe8dd12ab7fd7271a3962a70df3fe3da50cfadf705df2ee10140074b9aaaad298d808f94a968
b884ff3854db55c0024009aeb69afee1d3cb20c209915e9c9a69fa93cb2500bbd0e24e3073d4bf7f6b98e2ed5374fd07d5d8809c8cfabe6d9731fdadcc52cc316343447
39f2f3079c0e94ab020168247dac4efa47aecde38f455ff8d6493a0d6dce26dc0d6d87b043f35c36352409f13f3c0608f8639a4ed14894
Verification Error: 5, invalid signature -- PASSED
---
s = 540cf6640c35cd421b1e920bb28a4dc9d8456c882d04d0a018bdd5477684b093fa58458961f636b545367864cd891ebc8fc04ae1a11f98e27c352c578afea9ac97a
9ca8f99ec7273b2fba54ced727411ca4aa36fbdfe18d358afc0096f7790ec3ff9c03b38c1d4d6c1adcd1c9ceb53b7e1ee7389183ba26a71303d61d1139d7e3cd3ae34e1
36b40af38ca31b8d5dbccaeacd4ae354162784b20091c8672d9cd40b40a6a89d8e8c3e93184f38cf7ecd30fcaa37eac9eeb2f78c7bb881462899309faf71860d2995a72
15b62dfa49cf76fc24965cd37b00194d4206cc6bd563fabc90891dad55bb0db388c45e714c53ecda9c2e97696a1202dccc8b0e951e683f
Verification Error: 8 -- PASSED
---
s = 000029eadadd476f02272688da67ef67d5bd71f713dcca72f29feb3d3dd38c880ea6a5bfff3a4ddefb7e2cc2723a12b944f4d71210960663493adb6f7882856eb7531
8fd6be73ca4cd9a0b286846ffe89791f118d3a7239b1c29967311fd25b27d2579b199c3d01283c4071bb456d4b60f5e1e2fadbf4281db4584ae2107a4baafbab774ae
95775dc08234cd323931d52000478e449b08705083b1a9d00eb4bcf3914f650e4c54895dcd918c183f9965b8b092f803378df00cc8bd93fb8dc576e299c7ad0a9debf9e
3cc98ee57c669b8ef341c469ac159c57573321c31f500c7afaf2e5d71510ae9e480052ab6be0c27d120b3ca29b1fc8ad47e28c79c28bb98
Verification Error: 8 -- PASSED
---
Compatible Signature Verification! -- PASSED
---
RSASSA-PSS Random Testing.....No error found
! -- PASSED
CPU 사용 시간 = 186.4715초
jihyeondo@192 proj#5-1 %

```

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.