

Project #5

2018044893 이수정

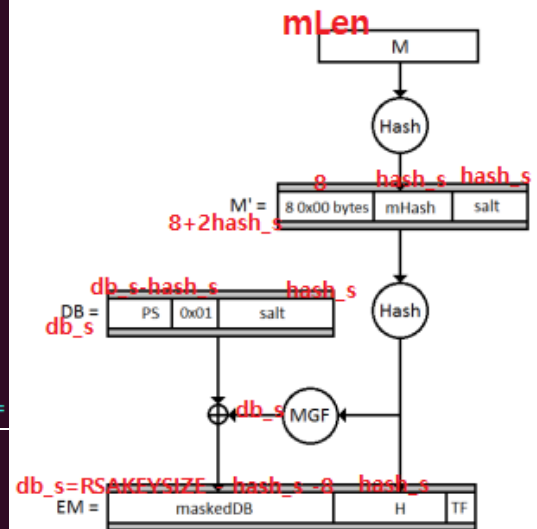
목차

1. 함수설명
2. 컴파일 과정
3. 실행 결과
4. 소감
5. 피드백 수정 내용
6. 전체 코드

<함수 설명>

- int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void *s)

```
170 * rsassa_pss_sign - RSA Signature Scheme with Appendix
171 */
172 int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void *s)
173 {
174     unsigned char mhash[HASH_S];
175     unsigned char salt[HASH_S];
176     unsigned char M_prime[8 + HASH_S * 2];
177
178     unsigned char H[HASH_S];
179     unsigned char mgf_H[DB_S];
180
181     unsigned char DB[DB_S];
182
183     unsigned char maskedDB[DB_S];
184     unsigned char EM[RSA_S];
185
186     uint64_t is_long = 1;
187     uint8_t bc = 0xbc;
188     is_long = (is_long << 60) - 1; // 2^61-1 (2^64bit =
189
190 #define HASH_S SHASIZE/8
191 #define RSA_S RSAKEYSIZE/8
192 #define DB_S (RSAKEYSIZE - SHASIZE - 8)/8
193 #define PAD2_S (DB_S - SHASIZE/8)
194
```



자료형은 *mgf에서 사용한 unsigned char을 이용하였다. unsigned char의 자료형 크기는 1byte이므로 이에 맞게 변수를 define하였다.

Padding2는 또한 자주사용 되어 추가로define하였다.

```
191 // --- error check ---
192 if(m >= n) return EM_MSG_OUT_OF_RANGE; //EM_MSG_OUT_OF_RANGE 메시지의 길이는 n보다 >
작아야함.
193
194 if(mLen > is_long && SHASIZE <= 256) return EM_MSG_TOO_LONG; //EM_MSG_TOO_LONG
195
196 if(RSA_S < HASH_S * 2 + 2) return EM_HASH_TOO_LONG; //EM_HASH_TOO_LONG
197
```

메시지 m이 n보다 크거나 같다면 메시지의 오류처리 해준다.

Steps:

1. If the length of M is greater than the input limitation for the hash function ($2^{61} - 1$ octets for SHA-1), output "inconsistent" and stop. (rfx8017.pdf 중)

공지에 올라온 [과제 #5] SHA 알고리즘 특성을 고려하여 SHASIZE가 256보다 작거나 같은 경우 메시지 사이즈가 2^{64} bit보다 작아야 하므로 2^{64} byte보다 작아야 한다.

3. If $emLen < hLen + sLen + 2$, output "inconsistent" and stop. (rfx8017.pdf 중)

이를 참고하여 hash길이가 너무 커 EM이 수용하지 못하는 경우의 예외를 설정하였다.

```
198 // --- M_prime ---
199 memset(M_prime, 0x00, 8); //padding1
200 sha(m, mLen, mhash); //mhash = hash(m)
201 *salt = arc4random_uniform(SHASIZE); //upperbound가 SHASIZE인 random # salt 생성
202 memcpy(M_prime + 8, mhash, HASH_S); //M_prime = pad1 || mhash
203 memcpy(M_prime + 8 + HASH_S, salt, HASH_S); // M_prime = pad1 || mhash || salt
204
```

M'의 padding1값을 memset을 이용하여 초기화 하였고 mhash = hash(m)을 함수 sha를 이용하였다.

Arc4random_uniform()은 bit사이즈를 이용하므로 SHASIZE를 넣어 난수를 생성해주었다.

생성된 mhash와 salt를 padding1이 추가된 M'에 추가해준다 (M' = padding1 || mhash || salt)

```
205 // --- H ---
206 sha(M_prime, 8 + HASH_S * 2, H); //H = hash(M_prime);
207
```

H = hash(M')이므로 mhash와 같이 함수 sha를 이용하였다.

```
208 // --- DB ---
209 memset(DB, 0x00, pad2_s); //padding2
210 DB[pad2_s - 1] = 0x01; // padding2 : 0x01
211
212 memcpy(DB + pad2_s, salt, hash_s); // DB = padding2 || salt
213
```

DB는 padding2와 salt로 만드는데 salt size에 따라 padding값이 유동적으로 바뀐다.

RSAKEYSIZE에서 H와 TF(0xbc)의 사이즈를 뺀 것이 DB의 사이즈가 되므로 $db_s = (RSAKEYSIZE - SHASIZE - 8)/8$ 로 설정하였고 padding은 DB사이즈에서 salt사이즈를 뺀 것이므로 $pad2_s = db_s - SHASIZE/8$ 로 설정하였다.

memset을 이용하여 pad2_s만큼 DB를 0으로 채워준 후 0의끝을 알려주는 0x01을 추가하였고 salt를 추가하였다. (DB = padding2 || salt)

```
215 // --- maskedDB ---
216 mgf(H, HASH_S, mgf_H, DB_S); //mgf_H = mgf(H)
217
218 for(int i = 0; i < DB_S; i++) maskedDB[i] = DB[i] ^ mgf_H[i]; //maskedDB = DB ^ mgf_H;
219
220 maskedDB[0] = 0x00; //MSB bit 0
221
```

maskedDB 는 DB XOR mgf(H)이므로 mgf 함수를 이용하여 $mgf_H = mgf(H)$ 를 생성하였다.

for문을 통해 DB와 mgf_H의 XOR연산을 해주었다.

MSB bit가 1일 때 0으로 바꿔줘야 해 MSB bit는 항상 0이므로 0인지 1인지 확인과정을 거치지 않고 0으로 설정하였다.

```

222 // --- EM ---
223
224 memcpy(EM, maskedDB, DB_S); //EM = masked_DB
225 memcpy(EM + DB_S, H, HASH_S); //EM = maskedDB || H
226 memcpy(EM + RSA_S - 1, &bc, 1); // EM = maskedDB || TF(0xbc)
227
228

```

EM = maskedDB || H || 0xbc 를 해주고 MSB는 maskedDB를 생성하는 과정에서 0으로 설정해두었다.

```

229 // --- error check ---
230 if(rsa_cipher(EM, d, n)) return EM_MSG_OUT_OF_RANGE; //EM cipher & EM_MSG_OUT_OF_
    RANGE
231
232 // ---
233 memcpy(s, EM, RSA_S); //s = EM
234 return 0;
235 }

```

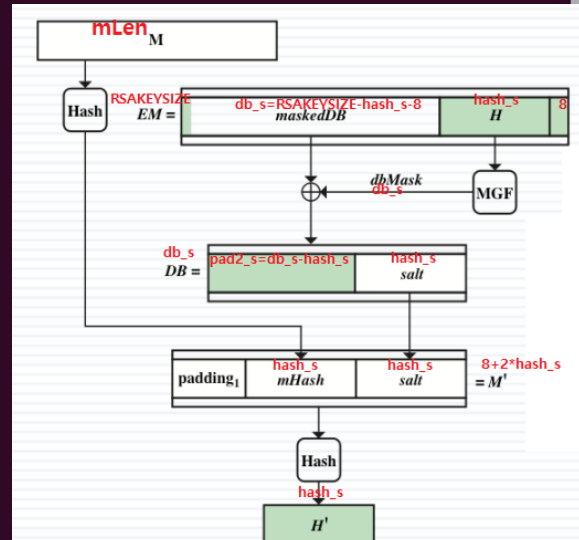
EM을 rsa_cipher함수로 암호화 해주고 EM >= n 인경우 rsa_cipher(EM, d, n)의 결과가 1이기 때문에 EM_MSG_OUT_OF_RANGE를 return해주고 오류없이 작동하였을 때 EM을 s에 저장한다.

- int rsassa_pss_verify(const void *m, size_t mLen, const void *e, const void *n, const void *s)

```

237 /*
238 * rsassa_pss_verify - RSA Signature Scheme with Appendix
239 */
240 int rsassa_pss_verify(const void *m, size_t mLen, const void *e, const void *n, const void *s)
241 {
242     unsigned char mhash[HASH_S];
243
244     unsigned char EM[RSA_S];
245     unsigned char maskedDB[DB_S];
246     unsigned char H[HASH_S];
247     unsigned char mgf_H[DB_S];
248
249     unsigned char DB[DB_S];
250     unsigned char salt[HASH_S];
251
252     unsigned char M_prime[8 + HASH_S * 2];
253
254     unsigned char H_prime[HASH_S];
255
256     uint64_t is_long = 1;
257     uint8_t bc = 0xbc;
258     is_long = (is_long << 60) - 1; //2^61-1
259
260     memcpy(EM, s, RSA_S); //EM = s
261

```



rsassa_pss_sign과 마찬가지로 변수를 설정하였고 확인 알고리즘에 맞게 H'을 추가하였다.

EM에 s값을 옮겨주었다.

```

262 // --- error check ---
263     if(m >= n)         return EM_MSG_OUT_OF_RANGE; //EM_MSG_OUT_OF_RANGE
264
265     if(mLen > is_long)   return EM_MSG_TOO_LONG; //EM_MSG_TOO_LONG
266
267     if(RSA_S < HASH_S * 2 + 2)    return EM_HASH_TOO_LONG; //EM_HASH_TOO_LONG
268
269     if(rsa_cipher(EM, e, n))       return EM_MSG_OUT_OF_RANGE; //EM_MSG_OUT_OF_RANGE
270
271     if(EM[RSAKEYSIZE/8 - 1] ^ bc)  return EM_INVALID_LAST; //EM_INVALID_LAST
272
273     if(EM[0] >> 7 & 1)            return EM_INVALID_INIT; //EM_INVALID_INIT
274

```

Sign과 마찬가지로 메시지 m이 n보다 크거나 같다면 메시지의 오류처리 해준다.

Steps:

1. If the length of M is greater than the input limitation for the hash function ($2^{61} - 1$ octets for SHA-1), output "inconsistent" and stop. (rfx8017.pdf 중)

공지에 올라온 [과제 #5] SHA 알고리즘 특성을 고려하여 SHASIZE가 256보다 작거나 같은 경우 메시지 사이즈가 2^{64} bit보다 작아야 하므로 2^{61} byte보다 작아야 한다.

3. If $emLen < hLen + sLen + 2$, output "inconsistent" and stop. (rfx8017.pdf 중)

이를 참고하여 hash길이가 너무 커 EM이 수용하지 못하는 경우의 예외를 설정하였다.

EM을 rsa_cipher함수로 암호화 해주고 $EM \geq n$ 인 경우 rsa_cipher(EM, d, n)의 결과가 1이기 때문에 EM_MSG_OUT_OF_RANGE를 return해준다.

4. If the rightmost octet of EM does not have hexadecimal value 0xbc, output "inconsistent" and stop.

복호화 된 EM의 마지막 바이트가 0xbc가 아닌 경우 EM_INVALID_LAST를 리턴 해준다.

6. If the leftmost $8emLen - emBits$ bits of the leftmost octet in maskedDB are not all equal to zero, output "inconsistent" and stop.

EM의 첫 번째 비트가 0이 아닐 경우 EM_INVALID_INIT을 리턴 해준다

```

275 // --- M ---
276     sha(m,mLen,mhash); //mhash = hash(m) step2
277

```

메시지 M에 대해 sha를 이용하여 해쉬를 해주었다.

```

278 // --- EM ---
279     memcpy(maskedDB, EM, DB_S ); // pick maskedDB from EM
280     memcpy(H, EM + DB_S, HASH_S); // pick H from EM
281
282     mgf(H, HASH_S , mgf_H, DB_S ); //mgf_H = mgf(H)
283

```

EM에서 db_s크기의 maskedDB와 hash_s크기의 H를 가져온 후 mgf함수를 사용하여 $mgf_H = mgf(H)$ 를 해준다.

```

287 // --- DB ---
288     for(int i = 0; i < db_s; i++)    DB[i] = maskedDB[i] ^ mgf_H[i];
289     memcpy(salt, DB + pad2_s , hash_s); //pick salt from DB
290
291     //-- check padding2 --
292     for(int j = 1; j < pad2_s - 1; j++){
293         if(DB[j] & 1)    return 6; //EM_INVALID_PD2
294     }
295     if(DB[pad2_s - 1] != 0x01)    return 6; //EM_INVALID_PD2
296

```

sign함수에서 $\text{maskedDB} = \text{DB} \oplus \text{mgf}(\text{H})$ 이므로 verify에서는 $\text{DB} = \text{maskedDB} \oplus \text{mgf}(\text{H})$ 를 연산을 해주었다.

DB에서 hash_s 크기의 salt값을 가져온다.

연산 된 DB에서 padding2를 확인하는 과정을 통해 EM_INVALID_PD2 오류를 확인한다. (padding2에 대한 내용은 뒤에서 추가 설명)

```

295 // --- M_prime ---
296     memset(M_prime, 0x00, 8); //M_prime = padding1
297     memcpy(M_prime + 8, mhash, HASH_S); //M_prime = padding1 || mhash
298     memcpy(M_prime + 8 + HASH_S, salt, HASH_S); //M_prime = padding || mhash || salt
299

```

검증을 위한 M을 해쉬한 mhash와 sign에서 가져온 salt을 합쳐 M'을 만든다.

```

300 //H_prime
301     sha(M_prime, 8 + HASH_S * 2, H_prime); // H_prime = hash(M_prime)
302

```

M'을 hash함수를 이용하여 H'을 만든다

```

303 // --- error7 check
304     if(memcmp(H, H_prime, HASH_S)!=0)    return EM_HASH_MISMATCH; //compare H, H_prime
305
306 // ---
307     return 0;
308 }

```

H와 H'을 비교하여 해시 값이 일치한지 확인하고 일치하지 않으면 EM_HASH_MISMATCH을 리턴해준다.

한번에 볼 수 있는 전체 코드는 보고서 마지막에서 볼 수 있습니다.

<컴파일 과정>

환경 : ubuntu-20.04.3-desktop-amd64(리눅스)

리눅스 환경을 고려하여 arc4random()을 사용하기 위해 2가지를 변경하였다

```
soo@soo-VirtualBox: ~/Documents/cryptography/hw5/project#5
soo@soo-VirtualBox:~/Documents/cryptography/hw5/project#5$ make clean
rm -rf *.o
rm -rf test
soo@soo-VirtualBox:~/Documents/cryptography/hw5/project#5$ make
gcc -Wall -c test.c
gcc -Wall -c rsa_pss.c
gcc -Wall -c sha2.c
gcc -Wall -o test test.o rsa_pss.o sha2.o -lbsd -lgmp
soo@soo-VirtualBox:~/Documents/cryptography/hw5/project#5$
```

1. test.c 함수의 #include<stdlib.h>를 #include<bsd/stdlib.h>로 변경하였다.
2. gcc 링크 시 -lbsd 옵션을 사용하기 위해서 Makefile에 -lbsd를 추가하였다.

<실행 결과물>

```
soo@soo-VirtualBox: ~/Documents/cryptography/hw5/project#5
e30a13584a0384919840e783281437c067aeea77b3df1a219e1011ec19d0233e675e37355bad8f6ef14ca4116571
178d2a6bdf87c4bda4ada852ae1714000caada628ea4ce191ca66b666734458155cfc48845501b47f092f9c53d2d
45c0a1ca7c8a6ea5e5cf435e661d6f3bed847f38110e006a8a3fb27e8a80036928c1cb85ecb3c69263d553c42e54
2df048099f96d53be2f2e7f3c5c41bd2334d7becfe1cf69858cd78dfbbf31f97bccb936d969e48aa3ee58fffc70f7
ab6c8d142a641ee6f9ad3d6b08105f2bf110595399e12b4d47e0f559
---
s = 5695f953f75ea0742a2d5003f844677bd5a75a234ad418c587051985912a8c228c32b88569f19a7a2530bc7e
7aa47d4580c88684c22539dd19463b56bcd58892673ed4e342c6790bc9e9a2cbb7e484ab6957c90f72ef4649513
b4deb9a472ececc9ff904c351e90175dc0a2223c35cff8076f2804e9f4f3dd00d0e88e62c6c9ca9e2b9949f77707
0b31bdb5842da28103305dc2d5f72e638e5c6bd4c851c580c7802fe03828673fe45357d674801747c7395927e52d
bd3026d98f26a3fb0c184d6a9279b1af64a4872201903dbefa7725ac5566e31079d63ebe96202adec0d5e941dd45
0f2bf51e290d696ce57316758fc5d1bda6f1abf656847f5803856595
Valid Signature! -- PASSED
---
s = 7a94710d59a65159833cbc22d1c1790d0e66efc6639f563b09e80e220485177e816f792e7fab36c4573fc00a
b53466b4df41dd3f1f4f18901c0f0873cea185441684607c75790c3d1dfcdefe73e8793b64a92a737803f47429ea
3e5201e4ccff979f7312f0739fda8df268931b7ab8d1616da6b2ae3ea84a76d5528933913f267eaa252283277d95
d9fe4cb1ce2fb9d610317504c9d3ec80dc53c86e2b0832e7a313fa84dab1b68e70671300ab51f9a72b204357d98c
65cca106125cc3e802c9e83022e93d7f0f7165dfd4f61a29339329fdb1a63e1b7751ab266171d010a5f407c7503d
feb0d964f5a16d34b700f90d3ca591c0b81bba61005f395bb8fc7059
Verification Error: 7, invalid signature -- PASSED
---
s = 529e9302295e81255f8491ce7cbe0ef05a8bb15d881aed1387be0d9588eb02e13f5c8d557d9be5c7c6856f6
57104764423639805d9e289ebfccc7f96551162303f69a75b03795da68a9fefb8d00c4f84e16b4fea8a5409171f7
1d9c25f98af4a6ef29ecc18aea6c2b0485750b6f89a6af11cef970c618521ba232afc998e0853ba157a9443cd9b5
153bef5a34e4b4e8f68ca5429f5d5ef56d717902422a1e2d1b263a772b6895c865be9397d0b59b789a3237b2225e
64e65d133c048da32d6d1d2a327366e488a83e70d39a7d1960f8ed8d6c10b78620cdaa8314769b88c25d43da69c3
733c58e1d19ac4e87b974212b2606b75e20190225c4680596bdf2699
Verification Error: 4 -- PASSED
---
s = 000063269c4c6dfe0a7ac5b43a0f03bd08340d6949f5386f1b1c7cf470eadbe79588fbd2bfb5def8f51f6830
1892ebf1cbcb0f913936ca042a14302b7e4809e087126095c4964798add8c366655e01d810e79852fb250a9f329
d9a7d57a4066dea80d399254f671ac7bd987862f0c0a88532c844239147ee7980436e8f5771738920ddf16b14bde
bb9e29a420dcd6da4edfe6659e17a62dc34c0ef919819a7e319039ca7f5d53ce51df6ef342de0f531095a0a79a2a
f8dd749d63f2e3680331084962a8ea9e0a72c0df8466600908819727058ae0c293836da56997e2528b30fb1eab0e
6f815f7b1c793f51bbc4e08cc25109fd815425b1366036e23b1b6501
Verification Error: 4 -- PASSED
---
Verification Error: 4, invalid signature -- PASSED
Compatible Signature Verification! -- PASSED
---
Random Testing.....No error found! -- PASSED
soo@soo-VirtualBox:~/Documents/cryptography/hw5/project#5$
```

No error found!를 확인하였다.

<소감/어려웠던점>

1. 자료형에 대한 고민

(수정 전)

```
165 int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void *s)
166 {
167     uint32_t *M, *pad1, *mhash, *salt, *DB, *ps, *EM, *maskedDB, *H, *hmgf;
168     uint32_t tmp_1 = 1;
169     uint32_t tmp_0 = 0;
170     if(rsa_cipher(s,d,n) == 1) return 1;//out of range
171
172     sha(m,mLen,&mhash);//mhash = hash(m)
173     salt = arc4random_uniform(SHA_SIZE);
174
175     mhash = (mhash << SHA_SIZE);
176     M = (tmp_0 << 2*SHA_SIZE);
177     M = (M | mhash | salt);//M_prime = pad1 || mhash || salt
178
179     DB = (tmp_0 << RSAKEYSIZE - 1) | (tmp_1 << SHA_SIZE) | salt;
180     //DB = ps || 0x01 || salt
181
182     sha(&M,8 + (SHA_SIZE/8) * 2, &H);
```

167,2-9

90%

(수정 후)

```
172 int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void *s)
173 {
174     unsigned char mhash[hash_s];
175     unsigned char salt[hash_s];
176     unsigned char M_prime[8+hash_s*2];
177
178     unsigned char H[hash_s];
179     unsigned char mgf_H[db_s];
180
181     unsigned char DB[db_s];
182
183     unsigned char maskedDB[db_s];
184     unsigned char EM[rsa_s];
185
186     uint64_t is_long = 1;
187     uint8_t bc = 0xbc;
188     is_long = (is_long << 60) - 1;//2^61-1
189
```

Arc4random을 통해 난수를 생성하기 때문에 uint32_t를 이용하여 연산하려 하였지만 비트 연산에서 어려움을 느껴 주소 값을 이용하여 계산할 수 있도록 코드를 수정하였다.

2. 배열 입력의 순서에 대한 고민

```
180     memset(M_prime,0,8+SHA_SIZE/8*2);//padding1
181     memcpy(M_prime + SHA_SIZE/8, mhash, SHA_SIZE/8); //M_prime = pad1 || mhash
182     memcpy(M_prime,salt, SHA_SIZE/8);
183
```

```
199     memset(M_prime, 0x00, 8);//padding1
200     sha(m, mLen, mhash); //mhash = hash(m)
201     *salt = arc4random_uniform(SHA_SIZE); //upperbound가 SHA_SIZE인 random # salt 생성
202     memcpy(M_prime + 8, mhash, hash_s); //M_prime = pad1 || mhash
203     memcpy(M_prime + 8 + hash_s, salt, hash_s); // M_prime = pad1 || mhash || salt
204
```

예를 들어 $M' = \text{padding} \parallel \text{mhash} \parallel \text{salt}$ 일 때 padding값이 $M'[0]$ 부터 적용되는지 $M'[8 + 2*\text{hash_s}]$ 부터 역으로 적용되는지 확신이 들지 않아 직접 확인해보았다.

3. verify함수에서 padding2 검증

전

```
294     //-- check padding2 --
295     for(int j=0;j<(DB_size - SHASIZE)/8 - 1;j++){
296         if(DB[j] & 1) return 6;
297     }
298     if(DB[(DB_size - SHASIZE)/8 - 1] != 0x01) return 6;
299
```

후

```
290
291     //-- check padding2 --
292     for(int j = 1;j < pad2_s - 1; j++){
293         if(DB[j] & 1) return 6;//EM_INVALID_PD2
294     }
295     if(DB[pad2_s - 1] != 0x01) return 6;//EM_INVALID_PD2
296
```

j = 0 부터 확인할 때 error가 발생하였다. EM에서 MSB를 강제로 0으로 설정하였기 때문에 verify에서 만들어낸 DB의 첫 번째 padding바이트가 0x00이 아닐 수도 있기 때문에 j = 1부터 검증하였고 padding2의 마지막 비트가 1인지 확인하였다.

4. M'생성 시 실수한 부분

```
276     memset(M_prime,0x00,8);
277     memcpy(M_prime + 8,mhash,mLen);
278     memcpy(M_prime + 8 + SHASIZE/8, salt, SHASIZE);
279
280     sha(M_prime,8 + SHASIZE/8 *2,H_prime);
281
294     memset(M_prime, 0x00, 8); //M_prime = padding1
295     memcpy(M_prime + 8, mhash, hash_s); //M_prime = padding1 || mhash
296     memcpy(M_prime + 8 + hash_s, salt, hash_s); //M_prime = padding || mhash || salt
297
298     //H_prime
299     sha(M_prime,8 + hash_s * 2, H_prime); // H_prime = hash(M_prime)
```

mhash를 M'에 추가해 줄 때 해쉬의 길이가 아닌 메시지의 길이를 붙여 넣어 오류가 발생하였고 알고리즘적으로 확인을 여러 번 했지만 찾지 못해 가장 오래동안 헤맨 부분이었다.

4번을 통해 전체 코드를 재정리할 수 있었고 공지내용 [과제 #5] 범하기 쉬운 오류: 확장성에 고려하여 RSA와 SHA가 변화되었을 때도 적용될 수 있도록 자주 쓰는 변수들을 define하였다.

이 과제를 진행하면서 익숙해진 비트연산을 이용하기 위해 자료형을 오랫동안 고민 해보았지만 헤더파일의 #include <string.h>라는 힌트를 통해 메모리 주소를 이용하는 연산으로 선회할 수 있었다.

변수의 실수 덕분에 전체적인 알고리즘을 오랫동안 다시 고민해보았고 주석처리도 평소보다 더욱 깔끔하게 할 수 있었다. 배열의 크기에 대해서도 수치적으로 깔끔하게 define할 수 있었다.

<피드백 반영>

- 피드백 내용 1. 틀린 것은 아니지만 전처리로 정의한 것은 일반 변수와 구분하기 위해 대문자를 사용하는 곳이 전세계 관례이다.

수정 전

```
20 #define hash_s SHASIZE/8
21 #define rsa_s RSAKEYSIZE/8
22 #define db_s (RSAKEYSIZE - SHASIZE - 8)/8
23 #define pad2_s (db_s - SHASIZE/8)
```

수정 후

```
20 #define HASH_S SHASIZE/8
21 #define RSA_S RSAKEYSIZE/8
22 #define DB_S (RSAKEYSIZE - SHASIZE - 8)/8
23 #define PAD2_S (DB_S - SHASIZE/8)
```

변수와 구분을 위해 피드백 내용을 반영하여 대문자로 변경하였다.

- 피드백 내용 2. 아래 EM_HASH_TOO_LONG 계산에 단위에서 오류가 있는 것 같다. 다시 한번 확인해라.

수정 전

```
195
196     if(RSAKEYSIZE < SHASIZE * 2 + 2)         return 3; //EM_HASH_TOO_LONG
197
266     if(RSAKEYSIZE < SHASIZE * 2 + 2)         return 3; //EM_HASH_TOO_LONG
```

수정 후

```
196     if(RSA_S < HASH_S * 2 + 2)         return EM_HASH_TOO_LONG; //EM_HASH_TOO_LONG
197
267     if(RSA_S < HASH_S * 2 + 2)         return EM_HASH_TOO_LONG; //EM_HASH_TOO_LONG
```

Rfc8017.pdf에서 제공한 3. If `emLen < hLen + sLen + 2`, output "inconsistent" and stop. 이 내용은 byte단위지만 비트 단위로 착각하여 계산단위 오류가 발생하였다.

- 피드백 내용 3. EM_HASH_TOO_LONG을 검증하는 과정에서 Padding2에서 공간이 부족하면 PS는 없어도 되지만 0x01은 필수로 처리한 것처럼 보인다. 우리는 salt의 길이를 알고 있기 때문에 PS와 0x01 둘 다 생략할 수 있는 것 아닌가? 그 문제에 대해서 너는 어떻게 생각하는가? 보고서에 네 의견을 서술하기 바란다.
- 질문의 요지는 공간이 부족하면 0x01도 생략할 수 있냐?였다. 가능한지 아님 불가능한지, 그 이유는 무엇인지. 잘 생각해보고 다시 답변해라. --- 나머지는 옳게 수정했다.

```
208 // --- DB ---
209     memset(DB, 0x00, pad2_s );//padding2
210     DB[pad2_s - 1] = 0x01; // padding2 : 0x01
211
212     memcpy(DB + pad2_s, salt, hash_s ); // DB = padding2 || salt
213
```

공간이 부족하면 0x01도 생략할 수 있는가?

→ 생략하면 안된다고 생각합니다.

그 이유는?

→ $RSA = padding2 \parallel salt \parallel H \parallel 0xbc$ 이고 여기서 $salt \parallel H \parallel 0xbc$ 의 크기는 $HASH_S * 2 + 1$ 입니다.

곰곰히 다시 생각해 본 결과 제 코드에서는 해쉬의 길이에 대한 에러를 처리하는 과정을 거쳤습니다.

```
if(RSA_S < HASH_S * 2 + 2) return EM_HASH_TOO_LONG; //EM_HASH_TOO_LONG
```

$RSA_S \geq HASH_S * 2 + 2$ 이기 때문에 padding2의 0x01이 존재할 수 있는 공간이 충분합니다.

0x01을 생략할 만큼 공간이 부족하려면 EM_HASH_TOO_LONG에서 처리되기 때문에 0x01은 생략하지 않아도 된다고 생각합니다.

- 피드백 내용 4. 상수를 피하려고 오류 메시지를 정의했는데, 그 사용의 의미가 퇴색됨.

return 3; //EM_HASH_TOO_LONG → return EM_HASH_TOO_LONG;

```
228 // --- error check ---
229     if(rsa_cipher(EM, d, n)) return EM_MSG_OUT_OF_RANGE; //EM cipher & EM_MSG_OUT_OF_
RANGE
230
191 // --- error check ---
192     if(m >= n) return EM_MSG_OUT_OF_RANGE; //EM_MSG_OUT_OF_RANGE 메세지의 길이는 n보다 >
작아야함.
193
194     if(mLen > is_long && SHASIZE <= 256) return EM_MSG_TOO_LONG; //EM_MSG_TOO_LONG
195
196     if(RSA_S < HASH_S * 2 + 2) return EM_HASH_TOO_LONG; //EM_HASH_TOO_LONG
197
261 // --- error check ---
262     if(m >= n) return EM_MSG_OUT_OF_RANGE; //EM_MSG_OUT_OF_RANGE
263
264     if(mLen > is_long) return EM_MSG_TOO_LONG; //EM_MSG_TOO_LONG
265
266     if(RSAKEYSIZE < SHASIZE * 2 + 2) return EM_HASH_TOO_LONG; //EM_HASH_TOO_LONG
267
268     if(rsa_cipher(EM, e, n)) return EM_MSG_OUT_OF_RANGE; //EM_MSG_OUT_OF_RANGE
269
270     if(EM[RSAKEYSIZE/8 - 1] ^ bc) return EM_INVALID_LAST; //EM_INVALID_LAST
271
272     if(EM[0] >> 7 & 1) return EM_INVALID_INIT; //EM_INVALID_INIT
273
288 //--- check padding2 --
289     if(PAD2_S > 0){
290         for(int j = 1; j < PAD2_S - 1; j++){
291             if(DB[j] & 1) return EM_INVALID_PD2; //EM_INVALID_PD2
292         }
293         if(DB[PAD2_S - 1] != 0x01) return EM_INVALID_PD2; //EM_INVALID_PD2
294     }
301 // --- error7 check
302     if(memcmp(H, H_prime, HASH_S) != 0) return EM_HASH_MISMATCH; //compare H, H_prime
303
```

공지 내용 [과제 #5] 범하기 쉬운 오류: 확장성에 위배된 내용으로 RSA와 SHA만 고려한 나머지 error메시지에 대한 확장성을 고려하지 못했다. 코드 전체의 오류 메시지를 수정하였다.

전체 코드 입니다.

```
172 int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void *s)
173 {
174     unsigned char mhash[HASH_S];
175     unsigned char salt[HASH_S];
176     unsigned char M_prime[8 + HASH_S * 2];
177
178     unsigned char H[HASH_S];
179     unsigned char mgf_H[DB_S];
180
181     unsigned char DB[DB_S];
182
183     unsigned char maskedDB[DB_S];
184     unsigned char EM[RSA_S];
185
186     uint64_t is_long = 1;
187     uint8_t bc = 0xbc;
188     is_long = (is_long << 60) - 1; // 2^61-1 (2^64bit = 2^61byte)
189
190
191 // --- error check ---
192     if(m >= n) return EM_MSG_OUT_OF_RANGE; // EM_MSG_OUT_OF_RANGE 메시지의 길이는 n보다 작아야함.
193
194     if(mLen > is_long && SHASIZE <= 256) return EM_MSG_TOO_LONG; // EM_MSG_TOO_LONG
195
196     if(RSA_S < HASH_S * 2 + 2) return EM_HASH_TOO_LONG; // EM_HASH_TOO_LONG
197
198 // --- M_prime ---
199     memset(M_prime, 0x00, 8); // padding1
200     sha(m, mLen, mhash); // mhash = hash(m)
201     *salt = arc4random_uniform(SHASIZE); // upperbound가 SHASIZE인 random # salt 생성
202     memcpy(M_prime + 8, mhash, HASH_S); // M_prime = pad1 || mhash
203     memcpy(M_prime + 8 + HASH_S, salt, HASH_S); // M_prime = pad1 || mhash || salt
204
205 // --- H ---
206     sha(M_prime, 8 + HASH_S * 2, H); // H = hash(M_prime);
207
208 // --- DB ---
209
210     memset(DB, 0x00, PAD2_S); // padding2
211     DB[PAD2_S - 1] = 0x01; // padding2 : 0x01
212
213     memcpy(DB + PAD2_S, salt, HASH_S); // DB = padding2 || salt
214
215 // --- maskedDB ---
216     mgf(H, HASH_S, mgf_H, DB_S); // mgf_H = mgf(H)
217
218     for(int i = 0; i < DB_S; i++) maskedDB[i] = DB[i] ^ mgf_H[i]; // maskedDB = DB ^ mgf_H;
219
220     maskedDB[0] = 0x00; // MSB bit 0
221
222 // --- EM ---
223
224     memcpy(EM, maskedDB, DB_S); // EM = masked_DB
225     memcpy(EM + DB_S, H, HASH_S); // EM = maskedDB || H
226     memcpy(EM + RSA_S - 1, &bc, 1); // EM = maskedDB || TF(0xbc)
227
228
229 // --- error check ---
230     if(rsa_cipher(EM, d, n)) return EM_MSG_OUT_OF_RANGE; // EM cipher & EM_MSG_OUT_OF_RANGE
231
232 // ---
233     memcpy(s, EM, RSA_S); // s = EM
234     return 0;
235 }
```

```

240 int rsassa_pss_verify(const void *m, size_t mLen, const void *e, const void *n, const void *s)
241 {
242     unsigned char mhash[HASH_S];
243
244     unsigned char EM[RSA_S];
245     unsigned char maskedDB[DB_S];
246     unsigned char H[HASH_S];
247     unsigned char mgf_H[DB_S];
248
249     unsigned char DB[DB_S];
250     unsigned char salt[HASH_S];
251
252     unsigned char M_prime[8 + HASH_S * 2];
253
254     unsigned char H_prime[HASH_S];
255
256     uint64_t is_long = 1;
257     uint8_t bc = 0xbc;
258     is_long = (is_long << 60) - 1; // 2^61-1
259
260     memcpy(EM, s, RSA_S); // EM = s
261
262     // --- error check ---
263     if(m >= n)         return EM_MSG_OUT_OF_RANGE; // EM_MSG_OUT_OF_RANGE
264
265     if(mLen > is_long)   return EM_MSG_TOO_LONG; // EM_MSG_TOO_LONG
266
267     if(RSA_S < HASH_S * 2 + 2)   return EM_HASH_TOO_LONG; // EM_HASH_TOO_LONG
268
269     if(rsa_cipher(EM, e, n))      return EM_MSG_OUT_OF_RANGE; // EM_MSG_OUT_OF_RANGE
270
271     if(EM[RSAKEYSIZE/8 - 1] ^ bc) return EM_INVALID_LAST; // EM_INVALID_LAST
272
273     if(EM[0] >> 7 & 1)          return EM_INVALID_INIT; // EM_INVALID_INIT
274
275     // --- M ---
276     sha(m, mLen, mhash); // mhash = hash(m) step2
277
278     // --- EM ---
279     memcpy(maskedDB, EM, DB_S); // pick maskedDB from EM
280     memcpy(H, EM + DB_S, HASH_S); // pick H from EM
281
282     mgf(H, HASH_S, mgf_H, DB_S); // mgf_H = mgf(H)
283
284     // --- DB ---
285     for(int i = 0; i < DB_S; i++) DB[i] = maskedDB[i] ^ mgf_H[i];
286     memcpy(salt, DB + PAD2_S, HASH_S); // pick salt from DB
287
288     //--- check padding2 ---
289     for(int j = 1; j < PAD2_S - 1; j++){
290         if(DB[j] & 1) return EM_INVALID_PD2; // EM_INVALID_PD2
291     }
292     if(DB[PAD2_S - 1] != 0x01) return EM_INVALID_PD2; // EM_INVALID_PD2
293
294     // --- M_prime ---
295     memset(M_prime, 0x00, 8); // M_prime = padding1
296     memcpy(M_prime + 8, mhash, HASH_S); // M_prime = padding1 || mhash
297     memcpy(M_prime + 8 + HASH_S, salt, HASH_S); // M_prime = padding || mhash || salt
298
299     // H_prime
300     sha(M_prime, 8 + HASH_S * 2, H_prime); // H_prime = hash(M_prime)
301
302     // --- error7 check
303     if(memcmp(H, H_prime, HASH_S) != 0) return EM_HASH_MISMATCH; // compare H, H_prime
304
305     // ---
306     return 0;
307 }

```