Project5 - 암호학

A3조 고동우 우경완 이지현 최성환

1. 함수설명

1) sel_shaFunc

사용할 sha2 함수를 선택해서 호출하는 함수이다. input으로 message, len, digest, sha2_ndx을 받고, sha2_ndx에 맞는 sha 함수를 호출한다. switch case문을 이용하였다.

2) sel_shaLen

sha2_ndx 함수의 output 길이를 return하는 함수이다. input으로 sha2_ndx를 받고 sha2_ndx에 맞는 output 길이를 return한다. switch case문을 이용하였다.

3) i2osp

int 값을 octet string으로 변환해주는 함수이다. input으로 x, xLen, digit을 받고, int x를 xLen 길이의 octet string으로 변환한 후, digit에 저장한다. 이 함수는 mgf 함수에서 사용된다.

4) mgf

mgfSeed를 maskLen 길이인 mask로 변환한 후 return하는 함수이다. input으로 mgfSeed, seedLen, maks, maskLen, sha2_ndx을 받는다. mgfStore 배열에 mgfSeed를 저장하고, counter만큼 for (int i = 0; i < counter; i++) 내부의 i2osp, sel_shaFun 함수를 반복한다. i2osp 함수를 이용해 i를 octet string으로 변환한 뒤 mgfStore 배열의 seedLen 길이 뒤에 저장한다. 그 후 sel_shaFunc 함수를 이용해 mgfStore 전체를 sha2 함수에 넣은 값을 t 배열에 저장한다. 반복문이 끝나면 t 배열을 mask 배열에 maskLen 길이만큼 저장한 후 mask를 return한다.

5) rsaes_oaep_encrypt

길이가 len 바이트인 메시지 m을 공개키 (e,n)으로 암호화한 결과를 c에 저장하는 함수이다.

1. length checking

mLen 길이가 RSAKEYSIZE/8 - 2*hlen -2보다 크면 msg too long 에러 메세지를 return한다.

2. EME-OAEP encoding

label을 sel_shaFunc 함수를 이용해 hlabel로 변환해준 후, db 배열에 hlabel, 0x00, 0x01, m을 memcpy를 이용해 순서대로 concat한다.

arc4random_buf 함수를 이용해 seed를 생성한 후, seed를 mgf 함수에 통과시킨 값을 dbMask에 저장한다.(mgf1)

maskedDB 배열에 db와 dbMask를 xor한 값을 저장한다.

maskedDB를 mgf 함수에 통과시킨 값을 seedMask에 저장한다.(mgf2)

em 배열에 0x00, maskedSeed, maskedDB를 순서대로 concat한다.

3. RSA encryption

rsa_cipher 함수를 이용해 em을 공개키 e, n으로 encrypt한다. 만약 rsa_cipher에서 1을 return한다면 msg out of range 에러 메세지를 return한다.

rsa_cipher에서 1을 return하지 않았다면 c에 em을 길이 RSAKEYSIZE/8만큼 저장해준 후 0을 return한다.

6) rsaes_oaep_decrypt

1. EM^e을 복호화 하여 EM을 구한다.

2. 입력 받은 라벨 label을 sha2_ndx에서 정한 해시 함수를 이용하여 해시값(lHash)을 구한다.

3. EM의 첫 옥텟이 "0x00" 인지 확인하고 아니면 PKCS_INITIAL_NONZERO에러를 반환한다.

4. EM을 0x00 + maskedSeed + maskedDB 세 부분으로 나눈다. maskedSeed는 첫번째 옥텟부터 hLen만큼, maskedDB는 maskedSeed 이후부터 (RSAKEYSIZE/8) - hLen - 1만큼이다.

5. maskedDB를 MGF에 넣어서 hLen길이 만큼 출력한 seedMask를 구한다. (seedMask = MGF(maskedDB, hLen))

6. seedMask XOR maskedSeed = seed를 구한다.

7. seed를 MGF에 넣어서 (RSAKEYSIZE/8) - hLen - 1길이 만큼 출력한 DBMask를 구한다. (DBMask = MGF(seed, (RSAKEYSIZE/8) - hLen - 1))

8. DBMask XOR maskedDB = DB를 구한다.

9. DBMask를 lHash' + PS + "0x01" + M 네 부분으로 나눈다.

10. lHash와 lHash'가 일치하는지 확인한다. 일치하지 않으면 PKCS_HASH_MISMATCH를 반환한다.

11. PS와 M를 구분하는 "0x01" 이 있는지 확인한다. 없으면PKCS_INVALID_PS를 반환한다.

12. M을 m에 저장한다.

7) rsassa_pss_sign

우선 메시지 길이를 검사를 해서 오류를 반환할지 결정한다. 검사를 통과하면 해시함수 번호(sha2_ndx)를 통해서 해시, 키, masked DB, PS의 길이를 결정지은 뒤 연산에 필요한 배열과 변수들을 미리 생성한다. sha2_ndx에 따라 메시지(m)을 해시함수를 통해 mHash에 저장하고, salt에 난수를 저장한 뒤, M'배열(mPrime)에 이어 붙인다. 이때, 두 값을 이어 붙이기 전에 첫 8바이트를 0으로 설정해준다. 이렇게 생성된 M'를 다시 해시함수를 통해 H에 저장하고, M'과 salt를 DB배열에 이어 붙인다. 이때도 두 값을 이어 붙이기 전에 해시 길이에 따른 변수(psSize)만큼 DB에 0을 설정해준다. masked DB(MDB)를 생성하기 위해서 H를 mgf함수를 통해 변환하고 이 값(MGF)와 DB를 xor 연산하고, 연산 결과값의 가장 왼쪽값이 1일경우 0으로 수정해준다. 마지막으로 MDB, H를 EM배열에 복사해서 연결시키고 마지막 1바이트를 '0xBC'로 저장한다. 이 값(EM)을 암호화하고, 결과값이 정상적일 경우에만 암호화 결과값을 저장하고자 하는 포인터에 저장해주고 최종적으로 암호화 함수의 반환값을 반환해준다.

8) rsassa_pss_verify

rsassa_pass_verify함수는 메시지 m에 대한 서명이 s가 맞는 지 검사하는 함수이다. em에서 추출한 h와 m'을 hash한 h'을 비교해서 값이 같으면 메시지 m에 대한 서명이 s인 것이고 값이 다르면 메시지 m에 대한 서명이 s가 아니다. 여기서 m'을 구하는 과정은 sign했을 때의 과정을 거꾸로 생각하면 된다. em에서 maskedDb와 h를 추출하고 maskedDb와 h를 mgf함수에 넣어서 나온 값 dbMask를 xor연산해서 db를 구한다. db에서 salt를 추출하고 m을 hash해서 mHash를 만든다. 이 salt와 mHash를 가지고 m'을 만들 수 있다. m'을 hash한 h'과 em에서 추출한 h를 비교하여 서명이 s가 맞는지 검사한다.

2. 컴파일 과정

```
root@LAPTOP-IFTR7CVN:/home/dongwoo/proj#5# make
gcc -Wall -O3 -c pkcs.c
gcc -o test test.o pkcs.o sha2.o -lgmp -lbsd
```

3. 실행 결과물 및 소감

```
root@LAPTOP-IFTR7CVN:/home/dongwoo/proj#5# ./test
e = 2664c7ed08145564d4c6cc2c243ba7a0787fa507a08d8b27644bd0c2fb6a39a147a580fa7b8991ee7eb35cae866a2deb6acd099e9f61c380c70c4823ee131122e449bfffa018700
0d47f14127a57bdddbb5fe31474e88cbc52cb188c7c51d9987d0f14c712a9f5f9f829528db65afbbb2e58dbd9e332667a00282f557f6f163233d7c0fa7b756f80e0e87d8c415e1d7335
ff6d9b531f4fb872605fec17fcbbc482fad19d1c04f72289d41ae9cd1e9fe8ea06889e0c1be987024934545ff9c5e757c92aee5e440f58eb3adef7bc7c3260bff2d90a650efc2676302
c34acd1a8ccd1aaa8628bab52297515ed92bc98bc21f39544b0458558d77263f7f2ba13a5bd
d = 284ff5421bcdd9d9148e4146db5801321c9bbd3d2b3636309da4c999da2e29131d93a646cb904b1f01ca8d7212d334039ba6463065a04c6ee67941a9a1c0f99660aa1b280d244c2
37772e92b548ae35818afee1a3cabf32f7ae9bbc2bbf16b974826a9fe88856406532f3dfd26e69389fa189f47ac5b30e7861c987aa17bf21bcad7332fd8acaf6f242b911718b8a038f7
5607a642f2cbede90a71414f4989c251d50ee31ad43bbcf75da6ab44782c4a240dea4521b141cb2c03725e242ff9198454f9b89adb3b7f1d1cf8bf58675a518a2d094036a6309c374b0
c82ed7dbf479291734053d01ca7ece17faa229edc85927508a1317db7f059a028a6c44a6bfd
n = c5a097f52c13b924d53287b8fd5d69759c6c002e9d824f4b332a722bc2f534dfbd4bc463e393675d8a155976ad398e573618042d5a2e7f810eb9aea1a2b680e6948caa8ccc9dc55
ba7036f4ee85e7547c7a0b857148a63ebc38162d04796e69cea277b646b997cff1cdf86c15abad205b4cc861be6fd5118d99028e0784e91d4ab18782f21ff8f5c8dba7b6514ad9a88db
f713ddb39a159d8c9e7b59f3963105be85ad905af4044f9eb7937b07fb9dc2669528d8471d3ef05b4989f5b9e398d53259eb4a55ad6c0613d46b866699bc69748d3f07c0b5b54b59238
e1bf3d2bc783623755210ef4f1e64fd32f15c07a9fc4c44119d5e6730f56141ee73c5179317
---
c = aaa413519a6125153cdd551df8e79b3658fb81aa48ddc53ca145f6d799a74aeed3d979e574aaf7717db0a7432ee836d904f1491772cd3318c6e09a20a3c24fa1c77008a2ef48692
274482dca4f0255db36477e53bd865bc832c53401fed0cd5712e50f033581c611c2c663d8248378bceaa089f66c470160fdfa9f7d5b3b58f9222215ec6a2916dba2347fc47187d6d4a4
da39755cfbf58be71d9cc28abbe3dcbabe440355e43c43dc12420d08501759c1d028b51afee10d78b68d273aec64e789204c57f9936a926b048969a241218eef59400d7a37b3c3d30f7
7fd3c397eb767a7c60f97f36093f176c69cb52b0a4bf98d3c4d72360613e8d1c7a5e0109798
m = 73616d706c6520646174610 0
msg = sample data, len = 12 -- PASSED
---
c = 6cd0cb3736bde6050b5806bd22e8bb22bc9bfacce2e6313a6a239587fe03386fc4dc95010f4b18cc27ed1b4e58aa8f166dd959180aeb2b860e11c88d53ffe0cb4b679befb5b9968
6cb49febbe5e880338ae138ccc4f71bc4348399dc60c303de8d4e23a3a3139fffcf82f75c2a04bc9f849f2334789702b353a711969c50d648e2f4b4c2a5218d8560451d7f4ead14a9a2
098e676918e1d98f52f2222be3e35e6723608c9911b2f7781a914d376a767726aa9c346ba3c51bf1bdedd1b40d371c4355f085c9a93ff7d7dd2b7df16bebb80800d60f61c89af367d76
01ebe7b6a7d38b3332a10538fd5aea5ba6cd4a9c9cd59131eefdd04ef743632109083 0aaae7
m = 6d617820646174610 06d6178 2b20646174610 0456d707479206d6573736176 5202d2d20504153534544 0a2d2d2d006b6f7265616e20706f657400ec9ca4eb8f99eca3bc0045696
e737465696e006d7367203d202573202d2d20504153534544 0a2d2d2d0a0052534145532d4f4145502052616e646f6d2045657374696e67004572726f723a206c656e203d20257a7520
2d2d204641494c45440a004e6f206572726f7220666f756e6421202d2d20504153534544 0a2d2d2d0073203d20
msg = max data, len = 190 -- PASSED
---
Encryption Error: 2, message is too long -- PASSED
---
c = 0180851d293861e507bcf5f6537c7c92f64e8a558288789 0d058fb22d2e2375c4cc4347bba1951c15d40a0fe612fe81eaeea3e1fe6b81e4762e1319346f5b1dd842e4128fbe9e12
a747375bc508193f5894a7fb0ed8d91fc3ada646d0ce04650a0555904a047b19d46c8294cb3276b04c1627cc0208fe67b0fbd84bfdff87b815a5da4d89086ebcea2809b3b5dcbd7f9ce
63c280839df71c277848db75cb054c2b1b500cba0d93ae4058ae1fde98c1005b40c50234447154624a5875b28fe6232d316cdb630c9499a2c738f639c7ea739945ac9f925042b466a29
b4b51b5c4889893268b350f7007c51ce714ede81434d37914a96531f308bfb0e4d79fbfaf20
Empty message -- PASSED
---
m = ec9ca4eb8f99eca3bc
msg = 윤동주, len = 9 -- PASSED
msg = 배움은 경험에서 얻는다. 경험하지 않은 것은 정보에 불과하다. - 아인슈타인 -- PASSED
---
RSAES-OAEP Random Testing.....................................................................................No error found! -- PASSED
```

```
---
s = 46f12aebce3c53280b39151d93353a2f17036d8a0e0e8395fb8467c219674e4608191175cd460a964be208ec0c49f5224b402a1da276c6e21a7507d7d5c6624c1ec1e1000b4dff7
addcb38c74700bbfdacfd0fbd520933383da166cd2d5b524215bf08c13509d85d06f91ae083e8122e48aef555e8162767a3e3d62752e1054aa679c0939ea2e7d92e041cafd000b15b81
2c04642f0a3c4cdcb02ca8cf35e05e860200240a4604a4786becb38b686450142bf61f282376fe921421f8c9dd7d9098b706f9ea8c1381927dae77162972d2fd4e63e26ff02ea91f3c3
f0541817a68a09f5da8c343c837ca7d310e0d119115da062d17af84eae05912da86865f00d7
Valid Signature! -- PASSED
---
Hash Input Error: 2 -- PASSED
---
s = 224308fb83856b35934168551acb63b765247e5c6e58e62520fe6bac83690da911a3922a8c9c7e6be33ef968250dd36e9344b3f5632fe44695666e981467e92ac5328af4940844b
4f3fcdbb4a807385aca4478c4af51f928188893aade9bece87edd11a4e3d5dcc7c2701c8df18504bae8d19405ae63cc9b7374029a94c62a300c03389db2c4df55cdcfe68c171e87e08a
db8b1fabe4860413e3b8d44230ed1f8083bf37a573994d0da8badd80ad3a9d719fb97bc0cb06cc70ff6c743f09e69682825b8976368514451950d350c0f44d4c52cafb68590e3dcbb39
fff4bf2e2cdcaf86e429e61e1a6a7f9f3c653111fcbdd5a6a63f6b7452dbd3f60aa02d54305
Verification Error: 5, invalid signature -- PASSED
---
s = 87827b30c0d02c4fdb57880b48ecc57380b9b6c4d90786b8e784ba315acdbf0db15ebb3ce1b48dee364f8862563279a8f20b6b8dc5d5dac8565535ef66d4be57d5dfeb7bc87cfde
a293b9d83d3dcae92556b902f9a3e5c1a25b0967ff0ef736c276f23d4d68c13f9e58cc82c10b4ce8e7da585dc79780701efd7ddfa3ff5cae36e6a826c2ff7429c7e98f689453edc143f
aa38c65b66721a39a9196d540259b359f5595ebff3d3f5d6a9dafd97d18554bcaab4f0471300e435fd81c7d733f940e3deb191d5da0f1fd8b35a4b78619c3c58d2a0671ceb34899f7c3
2b90e11daeca91424db19c2f7cbdca353135e2c1b8ab602485837ed45f5a1e85acd05734035
Verification Error: 8 -- PASSED
---
s = 00009380c467255072a15ade9623fb20ac719ae56785be236eefcd057b9d35c070693fe2fca221cb160a88e2be9ea64b7671429d891951670178c3dc03246cb2dd9b33afec586fc
ad25a8f20071fdb6686e29b47cda6f7614993c3fe151b3e028e16a1e3a090c8b24d0ef1d0a0fd2bab26fd4c65203df8b78f2d0253ce82a0a6f0039d9e170b970bdc856d60a9ad1c6206
4bfb79b51a5e8127b8dc775fcf4449322af9b0313c8f9e228203cafc29ef3a5c417107e3155cf6808f745da895a81e013d675538b1920f910b8d94a7cd6e5621be0d3d76b07077cfe77
bdbb98526584c82351e8b36955de96edafe61eced609ebc843d36fcbb01a2f5463fb6706833
Verification Error: 8 -- PASSED
---
Compatible Signature Verification! -- PASSED
---
RSASSA-PSS Random Testing.....................................................................................No error found! -- PASSED
CPU 사용시간 = 126.4347초
```

동우

rsassa_pass_verify 함수를 만들면서 rsa-pss 작동방식을 잘 이해할 수 있었다. rsa-pss를 verify를 할 수 있는 핵심적인 이유는 salt라는 동일한 값이 db와 m′에 있기 때문이라는 것을 알 수 있었다. verify할 때 처음에 알 수 있는 값은 em과 mhash뿐이다. verify과정은 em에 안에 들어 있는 h와 m′을 hash한 h′을 비교하면서 이루어진다. h를 구하는 것은 em안에 값을 추출하기 때문에 금방 구할 수 있다. h′을 구하는 것이 verify를 하는 데 가장 중요한 문제이다. h′은 m′을 hash한 것이기 때문에 h′을 구하기 위해서는 m′만 구하면 된다. m′은 mhash와 salt로 이루어져 있다. mhash는 이미 알고 있기 때문에 salt만 구하면 이 문제를 풀 수 있다. db에 salt가 들어있기 때문에 db를 mgf(h)와 maskedDb를 xor해서 구하면 verify를 할 수 있다. rsa-pss를 sign하는 과정에서 db와 m′에 salt라는 동일한 값을 포함했기 때문에 다음과 같이 verify를 할 수 있다는 것을 알게 되었다. 이외에도 octet string의 개념과 mgf 함수의 작동방식 등을 이해할 수 있는 좋은 프로젝트였다.

경완

OAEP의 원리는 중간고사를 준비하면서 공부했기 때문에 이해하는 데 어렵진 않았고, 구현하는 것 또한 큰 어려움이 없었다. 다만 OAEP와 PSS의 핵심 함수인 mgf를 정확히 이해하고 구현하는 데 오랜 시간이 걸려서 완전한 결과물을 내놓는데 많은 시간을 소요했었다. OAEP와 mask generating function이 어떻게 작동하는지 이해할 수 있는 좋은 기회였다.

지현

처음 코드를 작성할 때는 알고리즘이 굉장히 복잡해 보여 어디서부터 시작해야 할까 막막했었는데, 다 완성하고 보니 의외로 직관적인 부분이 많았다. 프로젝트를 진행하면서 특히 배열 길이를 정확하게 설정하는 것이 중요하다는 생각이 들었다. 코드 작성 도중 막혔던 부분을 떠올려 보면 대부분이 포인터 배열 오류였고, 문법적 오류를 다 고친 후에도 테스트 결과가 틀리게 나왔던 이유는 배열 길이 설정을 잘 못 했던 것이 원인인 경우가 많았다. 이번 과제에서 처음으로 교수님께서 pdf에 제공해 주신 테스트 벡터를 사용하여 전체 코드를 테스트하기 전에 프로그램이 올바르게 돌아가는지 확인하였는데, 프로그램을 빠르게 완성하는 데 큰 도움이 되었다. 어느 부분에서 문제가 발생하는지 파악한 것만으로도 코드를 수정하는 시간을 훨씬 단축할 수 있었다. 이번 프로젝트를 통해 디버깅의 중요성을 알 수 있었으며 앞으로도 테스트 벡터를 적극적으로 활용해야겠다고 생각하였다.

성환

이 함수를 처음 작성할 때 배열들을 동적할당하는 방법을 사용했지만 함수가 끝나기전 배열들의 메모리를 해제하는 과정에서 특정 배열(MDB)의 메모리를 해제할 경우 세그먼트 폴트 에러가 발생하는 문제가 있었다. 과제에서 주어진 조건들을 살펴보면 굳이 동적할당을 사용할 이유가 없다고 판단했기에 배열들을 미리 할당하는 방법으로 수정함으로써 해결했다. 또한 처음에는 모든 배열들을 생성하고 이를 사용해 다른 배열에 복사하는 과정을 모두 거치게 했지만 나중에 memset 함수를 사용하면 코드의 양을 줄일 수 있었다. 이 함수를 구현하면서 전자 서명에 대해 좀더 잘 알게 되었다고 생각한다.