Cryptography Project #3

소프트웨어학부

2018044720 석예림

1. rsa_pss.c 소스코드

```c
/*
 * Copyright 2020. Heekuck Oh, all rights reserved
 * 이 프로그램은 한양대학교 ERICA 소프트웨어학부 재학생을 위한 교육용으로 제작되었습니다.
 */
#include <stdlib.h>
#include <string.h>
#include <gmp.h>
#include "rsa_pss.h"

#if defined(SHA224)
void (*sha)(const unsigned char *, unsigned int, unsigned char *) = sha224;
#elif defined(SHA256)
void (*sha)(const unsigned char *, unsigned int, unsigned char *) = sha256;
#elif defined(SHA384)
void (*sha)(const unsigned char *, unsigned int, unsigned char *) = sha384;
#else
void (*sha)(const unsigned char *, unsigned int, unsigned char *) = sha512;
#endif

/*
 * Copyright 2020. Heekuck Oh, all rights reserved
 * rsa_generate_key() - generates RSA keys e, d and n in octet strings.
 * If mode = 0, then e = 65537 is used. Otherwise e will be randomly selected.
 * Carmichael's totient function Lambda(n) is used.
 */
void rsa_generate_key(void *_e, void *_d, void *_n, int mode)
{
    mpz_t p, q, lambda, e, d, n, gcd;
    gmp_randstate_t state;

    /*
     * Initialize mpz variables
     */
    mpz_inits(p, q, lambda, e, d, n, gcd, NULL);
    gmp_randinit_default(state);
    gmp_randseed_ui(state, arc4random());
    /*
     * Generate prime p and q such that 2^(RSAKEYSIZE-1) <= p*q < 2^RSAKEYSIZE
     */
    do {
        do {
```

```
            mpz_urandomb(p, state, RSAKEYSIZE/2);
        } while (mpz_probab_prime_p(p, 50) == 0);
        do {
            mpz_urandomb(q, state, RSAKEYSIZE/2);
        } while (mpz_probab_prime_p(q, 50) == 0);
        mpz_mul(n, p, q);
    } while (!mpz_tstbit(n, RSAKEYSIZE-1));
    /*
     * Generate e and d using Lambda(n)
     */
    mpz_sub_ui(p, p, 1);
    mpz_sub_ui(q, q, 1);
    mpz_lcm(lambda, p, q);
    if (mode == 0)
        mpz_set_ui(e, 65537);
    else do {
        mpz_urandomb(e, state, RSAKEYSIZE);
        mpz_gcd(gcd, e, lambda);
    } while (mpz_cmp(e, lambda) >= 0 || mpz_cmp_ui(gcd, 1) != 0);
    mpz_invert(d, e, lambda);
    /*
     * Convert mpz_t values into octet strings
     */
    mpz_export(_e, NULL, 1, (RSAKEYSIZE/8), 1, 0, e);
    mpz_export(_d, NULL, 1, (RSAKEYSIZE/8), 1, 0, d);
    mpz_export(_n, NULL, 1, (RSAKEYSIZE/8), 1, 0, n);
    /*
     * Free the space occupied by mpz variables
     */
    mpz_clears(p, q, lambda, e, d, n, gcd, NULL);
}

/*
 * Copyright 2020. Heekuck Oh, all rights reserved
 * rsa_cipher() - compute m^k mod n
 * If m >= n then returns EM_MSG_OUT_OF_RANGE, otherwise returns 0 for success.
 */
static int rsa_cipher(void *_m, const void *_k, const void *_n)
{
    mpz_t m, k, n;

    /*
     * Initialize mpz variables
     */
    mpz_inits(m, k, n, NULL);
    /*
     * Convert big-endian octets into mpz_t values
     */
    mpz_import(m, (RSAKEYSIZE/8), 1, 1, 1, 0, _m);
    mpz_import(k, (RSAKEYSIZE/8), 1, 1, 1, 0, _k);
    mpz_import(n, (RSAKEYSIZE/8), 1, 1, 1, 0, _n);
```

```
    /*
     * Compute m^k mod n
     */
    if (mpz_cmp(m, n) >= 0) {
        mpz_clears(m, k, n, NULL);
        return EM_MSG_OUT_OF_RANGE;
    }
    mpz_powm(m, m, k, n);
    /*
     * Convert mpz_t m into the octet string _m
     */
    mpz_export(_m, NULL, 1, (RSAKEYSIZE/8), 1, 0, m);
    /*
     * Free the space occupied by mpz variables
     */
    mpz_clears(m, k, n, NULL);
    return 0;
}

/*
 * Copyright 2020. Heekuck Oh, all rights reserved
 * A mask generation function based on a hash function
 */
static unsigned char *mgf(const unsigned char *mgfSeed, size_t seedLen, unsigned
char *mask, size_t maskLen)
{
    uint32_t i, count;
    size_t hLen;
    unsigned char *mgfIn, *p, *m;

    /*
     * Check if maskLen > 2^32*hLen
     */
    hLen = SHASIZE/8;
    if (maskLen > 0x0100000000*hLen)
        return NULL;
    /*
     * Generate octet string mask
     */
    if ((mgfIn = (unsigned char *)malloc(seedLen+4)) == NULL)
        return NULL;;
    memcpy(mgfIn, mgfSeed, seedLen);
    count = maskLen/hLen + (maskLen%hLen ? 1 : 0);
    if ((m = (unsigned char *)malloc(count*hLen)) == NULL)
        return NULL;
    p = (unsigned char *)&i;
    for (i = 0; i < count; i++) {
        mgfIn[seedLen] = p[3];
        mgfIn[seedLen+1] = p[2];
        mgfIn[seedLen+2] = p[1];
        mgfIn[seedLen+3] = p[0];
```

```
        (*sha)(mgfIn, seedLen+4, m+i*hLen);
    }
    /*
     * Copy the mask and free memory
     */
    memcpy(mask, m, maskLen);
    free(mgfIn); free(m);
    return mask;
}

/*
 * rsassa_pss_sign - RSA Signature Scheme with Appendix
 */
int rsassa_pss_sign(const void *m, size_t mLen, const void *d, const void *n, void
*s)
{
    unsigned char mHash[SHASIZE/8];
    unsigned char M[8+2*(SHASIZE/8)];
    unsigned char salt[SHASIZE/8];
    unsigned char H[SHASIZE/8];
    unsigned char DB[(RSAKEYSIZE/8)-(SHASIZE/8)-1];
    unsigned char maskedDB[(RSAKEYSIZE/8)-(SHASIZE/8)-1];
    unsigned char mMgf[(RSAKEYSIZE/8)-(SHASIZE/8)-1];
    unsigned char EM[(RSAKEYSIZE/8)];
    uint8_t msb = 0x01;
    uint8_t bc = 0xbc;

    if(((SHASIZE == 224) || (SHASIZE == 256)) && (mLen > 0x1fffffffffffffff)){
        return EM_MSG_TOO_LONG;
    }
    if(2*(SHASIZE/8)+2 > (RSAKEYSIZE/8)){
        return EM_HASH_TOO_LONG;
    }

    // M'
    for(int i = 0; i < 8; i++){
        M[i] = 0x00;
    }
    // mHash
    sha(m, mLen, mHash);
    memcpy(M+8, mHash, SHASIZE/8);
    // salt
    arc4random_buf(salt, SHASIZE/8);
    memcpy(M+8+SHASIZE/8, salt, SHASIZE/8);

    // H
    sha(M, 8+2*(SHASIZE/8), H);

    // DB
    for(int i = 0; i < (RSAKEYSIZE/8)-2*(SHASIZE/8)-1; i++){
        DB[i] = 0x00;
```

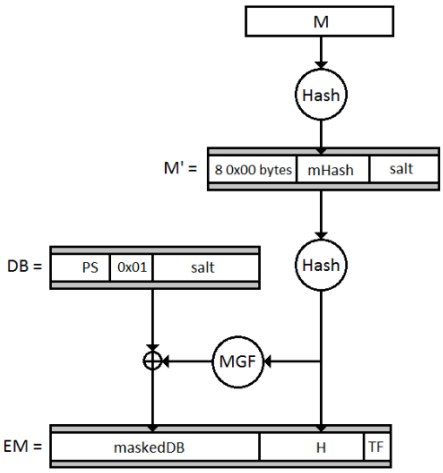
```
    }
    memcpy(DB+(RSAKEYSIZE/8)-2*(SHASIZE/8)-2, &msb, 1);
    memcpy(DB+(RSAKEYSIZE/8)-2*(SHASIZE/8)-1, salt, SHASIZE/8);

    // mgf(H)
    mgf(H, SHASIZE/8, mMgf, (RSAKEYSIZE/8)-SHASIZE/8-1);

    // maskedDB
    for(int i = 0; i < (RSAKEYSIZE/8)-(SHASIZE/8)-1; i++){
        maskedDB[i] = DB[i] ^ mMgf[i];
        EM[i] = maskedDB[i];
    }

    // EM
    if((EM[0]>>7) & 1){
        EM[0] = 0x00;
    }
    memcpy(EM+(RSAKEYSIZE/8)-SHASIZE/8-1, H, SHASIZE/8);
    memcpy(EM+(RSAKEYSIZE/8)-1, &bc, 1);

    // s
    if(rsa_cipher(EM, d, n)){
        return EM_MSG_OUT_OF_RANGE;
    }
    memcpy(s, EM, (RSAKEYSIZE/8));

    return 0;
}

/*
 * rsassa_pss_verify - RSA Signature Scheme with Appendix
 */
int rsassa_pss_verify(const void *m, size_t mLen, const void *e, const void *n,
const void *s)
{
    unsigned char M[8+(SHASIZE/8)*2];
    unsigned char salt[SHASIZE/8];
    unsigned char mHash[SHASIZE/8];
    unsigned char MHash[SHASIZE/8];
    unsigned char H[SHASIZE/8];
    unsigned char DB[(RSAKEYSIZE/8)-(SHASIZE/8)-1];
    unsigned char maskedDB[(RSAKEYSIZE/8)-(SHASIZE/8)-1];
    unsigned char EM[(RSAKEYSIZE/8)];
    unsigned char hMgf[SHASIZE/8];
    uint8_t bc = 0xbc;

    // EM
    memcpy(EM, s, (RSAKEYSIZE/8));
    if(rsa_cipher(EM, e, n)){
        return EM_MSG_OUT_OF_RANGE;
    }
```

```
    if((EM[0]>>7) & 1){
        return EM_INVALID_INIT;
    }
    if((EM[(RSAKEYSIZE/8)-1]) ^ bc){
        return EM_INVALID_LAST;
    }

    // maskedDB
    memcpy(maskedDB, EM, (RSAKEYSIZE/8)-(SHASIZE/8)-1);

    // H
    memcpy(H, EM+(RSAKEYSIZE/8)-(SHASIZE/8)-1, SHASIZE/8);

    // mgf(H)
    mgf(H, SHASIZE/8, hMgf, (RSAKEYSIZE/8)-(SHASIZE/8)-1);

    // DB
    DB[0] = 0x00;
    for(int i = 1; i < (RSAKEYSIZE/8)-(SHASIZE/8)-1; i++){
        DB[i] = maskedDB[i] ^ hMgf[i];
    }
    for(int i = 0; i < (RSAKEYSIZE/8)-2*(SHASIZE/8)-2; i++){
        if(DB[i] ^ 0x00){
            return EM_INVALID_PD2;
        }
    }
    if(DB[(RSAKEYSIZE/8)-2*(SHASIZE/8)-2] ^ 0x01){
        return EM_INVALID_PD2;
    }

    // salt
    memcpy(salt, DB+(RSAKEYSIZE/8)-2*(SHASIZE/8)-1, SHASIZE/8);

    // mHash
    sha(m, mLen, mHash);

    // M'
    for(int i = 0; i < 8; i++){
        M[i] = 0x00;
    }
    memcpy(M+8, mHash, SHASIZE/8);
    memcpy(M+8+SHASIZE/8, salt, SHASIZE/8);

    // Hash(M')
    sha(M, 8+2*(SHASIZE/8), MHash);

    for(int i = 0; i < SHASIZE/8 ; i++){
        if(H[i] ^ MHash[i]){
            return EM_HASH_MISMATCH;
        }
    }
```

```
    return 0;
}
```

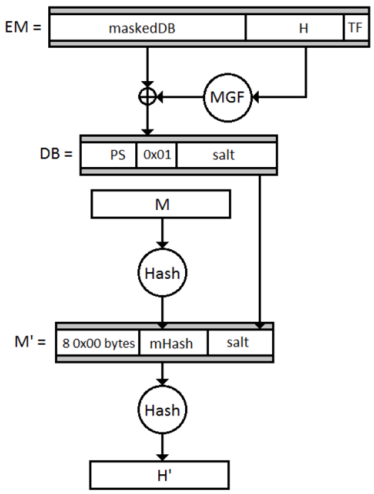
2. 코드 내 함수 설명

- rsassa_pss_sign 서명 과정



unsigned char 로 선언된 변수들은 바이트 단위이고, RSAKEYSIZE 와 SHASIZE 는 bit 단위이기 때문에 8 로 나눠 바이트 단위로 맞춰 변수 길이를 설정해주었다.
먼저, SHASIZE 가 224 또는 256 일 때에는 메시지 길이(mLen)가 64 비트를 넘어가지 않아야 하기 때문에 초과한다면 에러 처리(EM_MSG_TOO_LONG)를 해주었다.
EM 에 수용할 수 있는 해시의 길이인지 확인해 주기 위해서 maskedDB 의 길이와 H 의 길이와 TF 의 길이의 최소값이 EM 의 크기보다 작으면 된다. maskedDB 의 길이는 DB 의 길이와 같기 때문에 DB 를 사용하여 최소 길이를 계산해 보면 PS(패딩)값은 길이에 맞춰 채우기 때문에 최소 값은 0 이고, 0x01 : 1 바이트, salt 의 길이 : SAHSIZE/8 으로 최소길이는 SHASIZE/8+1 이다. H 의 길이는 SHASIZE/8 로 EM 의 최소 길이를 구해보면 2*(SHASIZE/8) + 2 이 된다. 이 값이 EM 의 길이인 RSAKEYSIZE/8 을 넘게 되면 수용할 수 없기 때문에 에러 처리 (EM_HASH_TOO _LONG) 를 해준다.
M'을 만들어 주기 위해 처음 8 바이트를 0x00 으로 채워주고 sha-2 계열의 함수를 사용하여 m 을 Hash 한 mHash 를 채워주고 arc4random_buf 를 사용하여 난수 salt 를 생성해 M'에 채워주었다.
DB 는 salt 와 그 앞에 0x01 을 채워주고 나머지를 모두 0x00 으로 채워준다.
만들어진 M'을 sha 함수를 사용해 Hash 해준 H 를 만들어 주었다. 만든 H 를 가지고 MGF 함수에 넣어 mMgf 를 생성해내고 이것을 DB 와 XOR 하여 EM 에 넣어준다.
EM 의 처음 비트가 1 이면 강제로 0 으로 바꾸어 줘야 하고, H 를 채우고, 마지막에 TF 1 바이트를 0xBC 로 채운다. 이렇게 만든 EM 을 가지고 EM^d mod n 을 계산하여 s 에 저장한다.

- rsassa_pss_verify 검증 과정



s^e mod n = EM 이므로, 계산하여 EM 에 저장해 주고, EM 의 첫번째 비트가 1 이면 EM_INVALID_INIT 에러 처리를 해 주고, EM 의 마지막이 0xbc 가 아니면 EM_INVALID_LAST 에러 처리를 해주고 종료시킨다. EM 에서 maskedDB 와 H 를 구해낸다. H 를 MGF 함수에 넣고 나온 값과 maskedDB 를 XOR 하여 DB 를 구해내고, 구한 DB 에서 salt 를 구한다. 서명과정에서 DB 와 H 를 XOR 했을 때 맨 처음 비트를 0 으로 강제로 바꾸어 주었기 때문에 검증과정에서 maskedDB 와 H 를 XOR 해 주면 첫 번째 비트가 1 이 나올 수 있다. 그래서 DB 의 첫 번째 비트를 0 으로 강제로 바꾸어 주고, DB 의 앞부분이 0x00 으로 채워져 있고, salt 앞 부분에는 0x01 로 채워져 있는지 확인하였다(EM_INVALID_PD2). DB 에서 구한 salt 와 m 을 Hash 한 후 구한 mHash 와 앞 8 바이트(0x00)을 가지고 M'을 구한다. 구한 M'을 Hash 하여 H 와 비교하여 검증한다(EM_HASH_MISMATCH).

3. 실행 결과

```
yerim ~
❯ cd Downloads/2020-2\ 암호학/프로젝트\ \#3
yerim ~/Downloads/2020-2 암호학/프로젝트 #3
❯ 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 -lgmp
yerim ~/Downloads/2020-2 암호학/프로젝트 #3
❯ ./test
e = 1ddb9137e8b478ac956ecfe19aa77a84415ad55444737f019d64c26bff41ad5f0c761e9d0eb7cd18847137a6941208f875dffc046bf65191c86900a82b2a6
ef27a4c634b4327b824c27aac33712f21450719ee22b580edc6599b47ae10fe5b30c5cee76b8003160bac39c301f09b46dfd843b35cefa9896fd6a68bc67dd3d2
e5c13f68a4472e8f54c732446ee4fbed050251efd1fd202e36f5aca1147aeaffd3387475f3864e2a726c3ae02be3a46adbe4b53207f0caafe2647053b42ae296b
465ecdfa081ae0fd6426c49d24ad8bedd06e311bb05b1ca3d2fd38df9f5d656697bcb86f54f9bc0b80cbb2d830b3ea2ea6b11292c48f06e82e0998ee272ea2839
d = 0a2cd393badbec540b005a1a50f93f9bfb14a954276bad762c427aed81660ac8eab76f51ec70c03edf41ccb2533b962a14b57c0f9b7ff3a778842f8436934
27d26fb74f2b9e1402a04988449f85a177b06f78b75282e835ab8097c7ea20e7a603855c5a7c62131e55519cc17cd30c2290053340abdfefa51501f74504344d3
49c13c769f5edeedbdc50535036f8a024bda46966901ae71315b46e45f84d3497d45dabdc66df020c59626d0bb02f58cbe8402f208f4ee8168072ff95304a764f
93066639a8924b419188e815c5c86c5d588ce6b91b72a64a48979f08bf16d24617c9fce04a927f8f0cd9499f562413f50aad94ceccbb8b41487609ecf1a95e9b7
n = 84c8196acd0e0792df2649bc32c0b955533ce00f303bd1ea81fe462b2c242661abf3f2ff6ea2fd4a881f23a8ac98c1cbe6408a327715e192af09513674773
5c4af334ebd94a79ff44e3e7d10f63222eddee731570e455306c47fa208742ea916b56353cdc7ba405e4ae2b45d5a75f52977ab32e12393a540254eaa1c8f354d
5f72c83c993c5c3ca3b1144c265d570753245a7595283d8a2710a3ba2c8d1821f02e19c2f7871ea33d35a053e15465578afa7cde96b5aa8a000ae2d75cdb0aa67
426f59808e262ecf935a4dcb86d4ec7f3110111db316345c824bb8b9a8e784b75e6c927376dabfd29e82a646f26d58f084be256b155824124253aade99a25d545
---
s = 0d3741b9e269dd95811ce35abc3174dc0c41b66419b1fb9ce522199e4af31b493665ce0adc84dfa1b0feb3790e3b1e26efcd97281ac695e23cd72d087a077
e0654499cd744d56d7d77955cf4e7c7ce437d13b3475ad4c70ff6050bf6db668af26f2e90ef58995de51bd653d4eb183591110db69a0586e020ab900d2d1a2e6f
c38bcd1778e90bfdf6615073877a08129976dcb642fda1597be026d6a5e85f2de9c7bda2259dafc836e3d4ee37802b1bced28b576ffac262330a6c087f4a2a91e
912f9d3baabf233ad9bd9f29de5df5f39839bda87258c510a17f3fd014dfe19123ff02a168f4ed34856844aedf9e3cb008a572fc7f4c31b1e3091f062334bcaa3
Valid Signature! -- PASSED
---
s = 68f9b81dd148c80b8de52ebf253d66f012d2f7a0dbf4a2353781230a6189e4a3208630706254090be11e3923362a58a3589b64e4c6b56465e0ae9da9eb361
038e4168907175cf5deacc7d3766a5a64bf25d1f1d6c3d2b6c2ce5d65b1206238a48ac30bc10441a828eb51d23f4357b4a6009030ee22b0e3d331dd572802730f
4bc4c88b17a7a611f5c8b0697c1c5b1e9ef44cc887a6f3b90c9b18a7200057eadb2837afd37859f4f333318ae36f5c60e8858322a9b50d80534b2b20c75cbccb6
8afa6dfaa342051f0ecb6f810d4b2bd8e5e1a9fd9cd92c3127bd02373973af8fe8a0e8d017a0fa457718525c1d8a5c4628005d36671d8b81297fd4507cf5390fe
Verification Error: 7, invalid signature -- PASSED
---
s = 16462110be77fd4f4082f2a31758602b41ca4f48940ea6da4bab0bbabcf31bf4f512a463ba3066f5ca6e40940af411f7fc89c643b32cb0c03933ad55961ba
586893d3087c00255baf4868865b2f3d730be2e0b6915e9a5990b11512be48f1f622749dadeb7f4964c02e752d4af1ef551b8d220a35ac245dba757921422d8f3
d29e1bdff77c2ea370fa1e2c1c4d1e3cd60040e77130c7f5a77027d401a84245b11e43fe6f1f475872008df01da735cfd03770b0a298acb512423383faefff179
31af536467f2a3256 8ddf186102556c8695a7f5c1804225e989fb70f445209c719b5b61ba928177038d4507aa93919e319fe99beb45ba5aa13128106bc4857d06
Verification Error: 4 -- PASSED
s = 0000e710d9c378ad8dc229d353ee27b7ac7ab18c65775eab2778db0f05d57886263013544505c9eb78cb85f71a878d11a3ec2778a5559aa61ef2d808a4396
d3803fbf1d61863eaa4c760702df94fab269057b2717a7e2a1a7f68b608ceb9116442c6d935e46758adf1549a31cdb09c5fa8dd82e03b5b6b1b5df0d2ae06c53d
d11edd69887d6ff95a3e4307af2abfb59fca1228babee2ff1ae865909ac3b4bef4263054975d3e2d782c00d413353280928aeeac7ea3a03625620fbbd1c53c8eb
b2c1ab1a59a01295fd5d56b795b34bf196b2f20518abd8a86f0194cfd4c642203825db977caf943a7fff5935729c2087ad24d232235ebb0bfe6db895a851848cf
Verification Error: 4 -- PASSED
Random Testing.....No error found! -- PASSED
yerim ~/Downloads/2020-2 암호학/프로젝트 #3
❯
```