암호학 HW4

2018037356 – 안동현

(아래 사진들은 코드의 캡쳐를 위해 주석을 지운 상태입니다. 본 파일에는 주석이 존재합니다)

#1 부가적 함수들

메인 함수들을 구성하기 위한 부가적인 함수들입니다.

1. static uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m);

```
9 static uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m)
0 {
1     uint64_t r1 = a % m;
2     uint64_t r2 = b % m;
3     return r1 < (m - r2) ? r1 + r2 : r1 - (m - r2);
4 }
5
```

이전 과제 3에서 작성하였던 mod_add함수이다.

오버플로가 나지 않게 조건을 구성하면서 a, b의 덧셈에 대한 모듈러 n 연산을 해주는 함수이다.

2. static uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m);

```
29 static uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m)
30 {
31     uint64_t r = 0;
32     while (b > 0){
33         if (b & 1) r = mod_add(r, a, m);
34         b = b >> 1;
35         a = mod_add(a,a,m);
36     }
37     return r;
38 }
39
```

이전 과제 3에서 작성하였던 mod_mul 함수이다.

오버플로가 나지 않고 빠르게 계산할 수 있도록 mod_add를 이용해서 double addition 알고리즘을 사용하였다.

3. static uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m);

```
 */
static uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m)
{
    uint64_t r = 1;
    while (b > 0){
        if (b & 1) r = mod_mul(r,a,m);
        b = b >> 1;
        a = mod_mul(a,a,m);
    }
    return r;
}
```

이전 과제 3에서 작성하였던 mod_pow 함수이다.

오버플로가 나지 않고 빠르게 계산할 수 있도록 mod_mul을 이용해서 square multiplication 알고리즘을 사용하였다.

4. static uint64_t gcd(uint64_t a, uint64_t b);

```
 */
static uint64_t gcd(uint64_t a, uint64_t b)
{
    uint64_t tem;
    while(b != 0){
        tem = b;
        b = a % b;
        a = tem;
    }
    return a;
}
```

이전 과제 1에서 작성하였던 유클리드 알고리즘이다. 과제 1과 달라진 점은 수의 범위가 32비트 int에서 unsigned 64비트 정수가 된 것이다. 그것 말고는 변화한게 없는 반복문을 사용한 유클리드 알고리즘이다.

5. static uint64_t mul_inv(uint64_t a, uint64_t m);

```
 */
static uint64_t mul_inv(uint64_t a, uint64_t m)
{
    uint64_t d0 = a, d1 = m;
    uint64_t x0 = 1, x1 = 0, q, tem, t;
    int isNx0 = 0, isNx1 = 0, isNt = 0;

    while (d1 > 1){
        q = d0 / d1;
        tem = d0 - q * d1;
        d0 = d1;
        d1 = tem;

        t = x1; isNt = isNx1;

        if (isNx0 != isNx1){
            x1 = x0 + q * x1;
            isNx1 = isNx0;
        }
        else{
            x1 = (x0 > q * x1) ? x0 - q * x1 : q * x1 - x0;
            isNx1 = (x0 > q * x1) ? 0 : 1;
        }
        x0 = t; isNx0 = isNt;
    }

    if (d1 == 1){
        return (isNx1 > 0 ? m - x1 : x1);
    }
    else
        return 0;
}
```

이전 과제 1에서 작성하였던 a의 모듈러 n에 대한 역원을 구하는 함수이다. unsigned 자료형을 사용해서 음수가 나올 수 없다는 점을 고려해 작성한 알고리즘이다.

6. static int miller_rabin(uint64_t n);

```
21 static int miller_rabin(uint64_t n)
22 {
23     uint64_t q = n - 1, tem, p, a_val;
24
25     if (n < 2 || (n != 2 && n % 2 == 0)) return COMPOSITE;
26
27     while (q % 2 == 0) q /= 2;
28
29     for (int i = 0; i < BASELEN; i++){
30         if((a_val = a[i]) == n) return PRIME;
31
32         tem = q;
33
34         if ((p = mod_pow(a_val, tem, n)) == 1) continue;
35
36         while(tem != n - 1 && p != n - 1){
37             p = mod_mul(p, p, n);
38             tem *= 2;
39         }
40
41         if (p != n - 1){
42             return COMPOSITE;
43         }
44     }
45
46     return PRIME;
47 }
```

이전 과제 3에서 작성하였던 밀러라빈 소수 판정 알고리즘이다. 오일러 정리를 이용해서 소수를 확률적으로 판단해주는 함수이지만, 64비트 자료형 내에서는 확정적으로 판단이 가능하기에 그것을 이용하여 소수판정을 해주는 알고리즘이다.

#2 메인 함수들

1. void mRSA_generate_key(uint64_t *e, uint64_t *d, uint64_t *n);

```
53  */
54 void mRSA_generate_key(uint64_t *e, uint64_t *d, uint64_t *n)
55 {
56     uint64_t p, q, s, l;
57
58     while (1){
59         p = arc4random(); p |= 0x80000001;
60         if (miller_rabin(p) != PRIME) continue;
61         q = arc4random(); q |= 0x80000001;
62         if (miller_rabin(q) != PRIME) continue;
63         s = p * q;
64         if (s < 0x8000000000000000) continue;
65
66         break;
67     }
68     *n = s;
69     l = ((p-1)*(q-1)) / gcd(p-1,q-1);
70     for (int i = 2; i < l; i++){
71         if (gcd(i,l) == 1){
72             *e = i;
73             break;
74         }
75     }
76     *d = mul_inv(*e, l);
77 }
```

공개키 PU{e, n} 과, 개인키 PR{d, n} 을 구하는 함수이다. 먼저 두 소수 p와 q의 곱으로 n을 만들기 위해서 p와 q를 랜덤으로 뽑는다. 이때 n은 무조건 2^63 <= n < 2^64 의 조건을 만족해야 하기에 여러번 뽑아보면서 조건을 만족하는지 체크해야 한다.

안전하게 랜덤으로 뽑기 위해서 32비트 정수를 리턴해주는 arc4random(); 함수를 이용해준다.

```
p = arc4random(); p |= 0x80000001;
if (miller_rabin(p) != PRIME) continue;
q = arc4random(); q |= 0x80000001;
if (miller_rabin(q) != PRIME) continue;
```

이렇게 각 p와 q를 뽑아서 두 수의 맨 왼쪽과 오른쪽 비트를 1로 바꾸는데 이는 해당 수를 홀수로 만들어주는 것의 의미하고 그 편이 유리하다. 이유는 2가 아닌 짝수는 소수가 아니기 때문이다.

두번째로 그냥 p와 q를 구하고 계속 소수인지 판정해보고 소수이면 곱해봤을 때 해당 수가 2^63 이상인지를 판단해야 하는데, 맨 왼쪽 비트를 1로 바꾸지 않고 계산을 했을 때 그 속도가 굉장히 느렸다. p와 q는 반드시 32비트일 필요는 없지만, 둘 다 32비트로 했을 때 조건에 맞는 p와 q를 찾는 속도가 비약적으로 빨라짐을 확인할 수 있었다.

따라서 두 수의 맨 왼쪽 비트를 1로 만들었다. 더군다나 p와 q가 비슷한 길이일수록 안정성이 높아진다고 하니 여러모로 이득이 많을 거라고 판단했다.

32비트 정수는 최대 2^32 - 1이고 (2^32 – 1) * (2^32 – 1) = 2^64 – 1이니 p*q는 항상 2^64 보다 작아서 오버플로를 걱정할 필요도 없다.

따라서 이렇게 계산해준 후

```
s = p * q;
if (s < 0x8000000000000000) continue;
```

2^63 보다 크다면 해당 조건을 넘어가 준다.

```
*n = s;
l = ((p-1)*(q-1)) / gcd(p-1,q-1);
for (int i = 2; i < l; i++){
    if (gcd(i,l) == 1){
        *e = i;
        break;
    }
}
*d = mul_inv(*e, l);
```

이후에 n값은 계산해준 s값으로 넣고, 카마이클 함수를 통해서 $\lambda(n)$를 구해준다.

이는 p – 1과 q – 1의 최소공배수이다.

이후 1 < e < $\lambda(n)$ 조건에 맞고, gcd(e, $\lambda(n)$) == 1 즉 서로소를 만족하는 e를 찾기 위해서 2 부터 $\lambda(n)$까지 반복문을 돌려준다. e를 찾았다면 반복문을 멈추고 e의 모듈러 $\lambda(n)$에 대한 역원인 d를 구해주는 것으로

e, d, n값을 모두 구해줄 수 있다.

이렇게 구해진 (e,n)은 공개키로 암호화에 쓰이고 (d,n)은 개인키로 복호화에 쓰인다.(반대도 가능)

2. int mRSA_cipher(uint64_t *m, uint64_t k, uint64_t n);

```
34 int mRSA_cipher(uint64_t *m, uint64_t k, uint64_t n)
35 {
36     if (*m >= n) return 1;
37     *m = mod_pow(*m, k, n);
38     return 0;
39 }
```

이제 암호화와 복호화인 m^k mod n을 해주는 함수를 구현한다.

m은 무조건 n보다 작아야 하기에 만약 더 크다면 난수로 뽑은 m이 너무 크다며 1을 리턴해주고, 그렇지 않다면 mod_pow로 m^k mod n를 계산하고 0을 리턴해준다.

#3 실행 결과

```
wollong@wollong-virtual-machine:~/proj#4$ vi mRSA.c
wollong@wollong-virtual-machine:~/proj#4$ make
gcc -Wall -O3 -c mRSA.c
gcc -o test test.o mRSA.o  -lbsd
wollong@wollong-virtual-machine:~/proj#4$ ./test
```

```
wollong@wollong-virtual-machine:~/proj#4$ ./test
e = 0000000000000007
d = 01478a63e2c74307
n = 8f4c8bb4b64600c1
m = 0, c = 0, v = 0
m = 1, c = 1, v = 1
m = 2, c = 128, v = 2
m = 3, c = 2187, v = 3
m = 4, c = 16384, v = 4
m = 5, c = 78125, v = 5
m = 6, c = 279936, v = 6
m = 7, c = 823543, v = 7
m = 8, c = 2097152, v = 8
m = 9, c = 4782969, v = 9
m = 10, c = 10000000, v = 10
m = 11, c = 19487171, v = 11
m = 12, c = 35831808, v = 12
m = 13, c = 62748517, v = 13
m = 14, c = 105413504, v = 14
m = 15, c = 170859375, v = 15
m = 16, c = 268435456, v = 16
m = 17, c = 410338673, v = 17
m = 18, c = 612220032, v = 18
m = 19, c = 893871739, v = 19
e = 000000000000000b
d = 0a68c6d91c627e55
n = e50116aa550d0e35
m = 9e2d14b5d6402b18, c = b7f5e1d5bd88991a, v = 9e2d14b5d6402b18
m = ab02f955fddc4690, c = 10cd6e9dcf6e262a, v = ab02f955fddc4690
m = 45887c6168074ddb, c = 2fe64b3d7323cb7c, v = 45887c6168074ddb
m = abb84f5e9fc3d48f, c = 7aa48221f6ebc3af, v = abb84f5e9fc3d48f
m = 38802d2bf4016be3, c = 26fc958c98241f6d, v = 38802d2bf4016be3
m = dd1c5e22a504ae0b, c = defef86d896074c5, v = dd1c5e22a504ae0b
m = 3f136346ac22008c, c = 779ac0e0b461e786, v = 3f136346ac22008c
m = 3c009f5d4966bd5e, c = 8d7b57a9f083bebc, v = 3c009f5d4966bd5e
m = fe4b6db8d402946d, m may be too big
m = 6c83aa77e6df3998, c = 609d9c2ee556a03c, v = 6c83aa77e6df3998
m = f9827ee7c83d2c0a, m may be too big
m = fa1789e5e9a3abab, m may be too big
m = 4f9e320b32a998cb, c = 906e69394527578b, v = 4f9e320b32a998cb
m = d1beda5f6b37fe93, c = 3e095a49fd0bea6f, v = d1beda5f6b37fe93
m = eb94cea068c5c459, m may be too big
m = 715d1fdafddc17f6, c = bbd429fa459d4f12, v = 715d1fdafddc17f6
m = 186d4fe2701248ac, c = 6968cfc07633417e, v = 186d4fe2701248ac
m = fcdcb86d07636b25, m may be too big
m = 66e95f601ef848e1, c = 826f10193899ed40, v = 66e95f601ef848e1
m = 3370929688d3570f, c = 5ea30eee60011da0, v = 3370929688d3570f
Random testing.................PASSED
wollong@wollong-virtual-machine:~/proj#4$
```

문제 없이 잘 출력되는 것을 확인할 수 있다. 예상 결과와도 비슷하고, e의 값만 조금 다른데

나의 알고리즘이 2 -> $\lambda(n)$ 까지의 수 중에서 e를 순차적으로 찾아보기에 값이 조금 작다는 것을 확인할 수 있다.

#4 느낀점

과제를 진행하면서 어려웠던 점은 p와 q를 랜덤으로 어떻게 뽑을까? 였었다.

p 와 q는 소수이고, p * q == n이면서 n의 길이는 64비트를 만족해야 하니, 조금 까다로웠다. 우려했던 점은 랜덤으로 뽑는 것이니 운이 정말 안 좋다면 프로그램이 영원히 돌지 않을까? 하는 걱정이였으나, 이건 로또를 맞는 것 그 이상으로 어마어마하게 확률이 낮아서 교수님께서 신경쓰지 말라고 하셔서 그냥 최대한 빠르게 조건을 만족하는 p와 q를 찾는데에 집중했다.

그것 외로는 어려운 개념이 있지도 않았고, 모두 이전 과제에서 진행했던 함수들이라서 어렵지 않게 과제를 마무리할 수 있었다.