암호학

Project #4

소프트웨어학부
2017012251
윤영훈

1. source code

```
CC=gcc
CFLAGS=-Wall

all: test.o mRSA.o
	$(CC) $(CFLAGS) -o test test.o mRSA.o -lbsd

test.o: test.c mRSA.h
	$(CC) $(CFLAGS) -c test.c

mRSA.o: mRSA.c mRSA.h
	$(CC) $(CFLAGS) -c mRSA.c

clean:
	rm -rf *.o
	rm -rf test
```

makefile -lbsd 추가

```
#include <stdlib.h>
#include "mRSA.h"

#include <bsd/stdlib.h>
```

mRSA.c #include <bsd/stdlib.h> 추가

```
#include <stdio.h>
#include <stdlib.h>
#include "mRSA.h"

#include <bsd/stdlib.h>
```

test.c #include <bsd/stdlib.h> 추가

source code를 설명하기에 앞서 주어진 makefile과 test.c, mRSA.c로는 linux 환경에서 arc4random 함수를 정상적으로 사용할 수 없으므로 다음과 같이 변경하였다.

① 지역 함수

```
uint64_t gcd(uint64_t a, uint64_t b){
    uint64_t temp = 0, result = 0;

    while (a != 0 && b != 0){
        temp = a;
        a = b;
        b = temp % b;
    }

    if (a == 0)
        result = b;
    else
        result = a;

    return result;
}
```

gcd

```
uint64_t mul_inv(uint64_t a, uint64_t m){
    uint64_t d0 = a, d1 = m;
    uint64_t x0 = 1, x1 = 0;

    uint64_t q = d0 / d1;
    uint64_t d2 = d0 - q * d1;
    uint64_t x2 = x0 - q * x1;

    while (d2 > 1){
        q = d0 / d1;
        d2 = d0 - q * d1;
        x2 = x0 - q * x1;

        d0 = d1;
        d1 = d2;
        x0 = x1;
        x1 = x2;
    }

    if (d2 == 1)
        return (x2 > (uint64_t)1<<63 ? x2+m : x2);
    else
        return 0;
}
```

mul_inv

```
uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m)
{
    while (a >= m || b >= m){
        if (a >= m) a -= m;
        if (b >= m) b -= m;
    }

    return ((a >= m-b) ? a-(m-b) : a+b);
}
```

mod_add

```
uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m)
{
    uint64_t r=0;

    while (b > 0){
        if (b & 1) r = mod_add(r, a, m);
        b = b >> 1;
        a = mod_add(a, a, m);
    }

    return r;
}
```

mod_mul

```c
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;
}
```

mod_pow

```c
int miller_rabin(uint64_t n)
{
    if (n % 2 == 0 && n != 2) return COMPOSITE;

    int k = 0;
    uint64_t q = n-1;

    while ((q % 2) == 0){
        q /= 2;
        k++;
    }

    for (int i=0; i<ALEN && a[i] < n-1; i++){
        uint64_t x = mod_pow(a[i], q, n);
        int count = 0;

        if (x == 1) continue;

        for (int j = 0; j < k; j++){
            if (mod_pow(x, 1 << j, n) == n-1){
                count++;
                break;
            }
        }

        if (count == 0) return COMPOSITE;
    }
    return PRIME;
}
```

miller_rabin

이전 프로젝트들에서 구현했던 gcd, mul_inv, mod_add, mod_mul, mod_pow, miller_rabin 총 6개의 지역 함수들을 사용했다. 이후 mRSA.h에서 지역 함수들 선언 및 PRIME, ALEN 등 필요한 변수를 define했다.

```c
#ifndef mRSA_H
#define mRSA_H

#include <stdint.h>

#define PRIME 1
#define COMPOSITE 0
#define MINIMUM_N 0x8000000000000000
#define ALEN 12

void mRSA_generate_key(uint64_t *e, uint64_t *d, uint64_t *n);
int mRSA_cipher(uint64_t *m, uint64_t k, uint64_t n);

uint64_t gcd(uint64_t a, uint64_t b);
uint64_t mul_inv(uint64_t a, uint64_t m);
uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m);
uint64_t mod_sub(uint64_t a, uint64_t b, uint64_t m);
uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m);
uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m);
int miller_rabin(uint64_t n);

#endif
```

mRSA.h

② mRSA_generate_key

```
16  void mRSA_generate_key(uint64_t *e, uint64_t *d, uint64_t *n)
17  {
18      uint64_t p, q;
19      uint64_t lambda_n;
20
21      uint64_t x = arc4random_uniform(0x7fffffff) + 0x80000000;
22      int num = 0;
23
24      while(1){
25          if (miller_rabin(x) && !num){
26              p = x;
27              x = arc4random_uniform(0x7fffffff) + 0x80000000;
28              num++;
29          }
30
31          else if (miller_rabin(x) && num){
32              q = x;
33              if (p * q > MINIMUM_N){
34                  *n = p * q;
35                  break;
36              }
37              else {
38                  num = 0;
39                  x = arc4random_uniform(0x7fffffff) + 0x80000000;
40              }
41          }
42          x++;
43      }
```

mRSA_generate_key 1

```
45      lambda_n = (p-1) * (q-1) / gcd(p-1, q-1);
46
47      // e, d
48
49      uint64_t random1, random2, i;
50
51      while (1){
52          random1 = arc4random_uniform(p-1);
53          random2 = arc4random_uniform(q-1);
54
55          i = random1 * random2 / gcd(p-1, q-1);
56
57          if (gcd(i, lambda_n) == 1){
58              *d = mul_inv(i, lambda_n);
59              if (*d == 0) continue;
60              else {
61                  *e = i;
62                  break;
63              }
64          }
65      }
66  }
```

mRSA_generate_key 2

p, q를 구하기 위해 첫 번째 while loop를 수행한다. 이때 랜덤한 32bit 정수 x를 arc4random_uniform을 통해 선언한다. arc4random_uniform은 upper bound 미만의 난수를 return하므로 arc4random_uniform(0x7fffffff)은 0 ~ $2^{31}-1$ 사이의 난수를 return한다. 이때 맨 앞 bit가 0이 아님을 보장하기 위해 0x80000000을 더해 2^{31} ~ $2^{32}-1$ 사이의 난수를 return한다. 이후 x를 1씩 증가시키며 miller_rabin 알고리즘을 통해 소수 여부를 확인하고 소수가 발견되면 p에 대입한 뒤 다시 x를 랜덤하게 재설정하고 동일한 loop를 반복하여 소수가 발견되면 q에 대입한다. 최종적으로 p와 q는 2^{31} ~ $2^{32}-1$ 사이의 소수로 설정된다.

이후 $\Phi(n)$ 대신 $\lambda(n)$을 사용하기 위해 lambda_n를 선언한다. 이후 e, d를 구하기 위해 두 번째 while loop를 수행한다. 이때 랜덤한 64bit 정수 i를 구하기 위해 랜덤한 32bit 정수 random1, random2를 서로 곱해주어 64bit bit 정수를 만들고, 이때 혹시라도 값이 lambda_n보다 커지지 않게 하기 위해 gcd(p-1, q-1)을 나누어 최종적으로 i를 선언한다. 이때 i와 lambda_n이 서로소이고 그때 e의 inverse인 d가 존재한다면, 해당 i로 e, d 값을 구한다. i와 lambda_n이 서로소가 아니거나 e의 inverse인 d가 존재하지 않는다면 다시 랜덤한 64bit 정수 i를 구하여 loop를 반복한다.

③ mRSA_cipher

```
72  int mRSA_cipher(uint64_t *m, uint64_t k, uint64_t n)
73  {
74      *m = mod_pow(*m, k, n);
75
76      if (*m >= n) return 1;
77      else return 0;
78  }
```

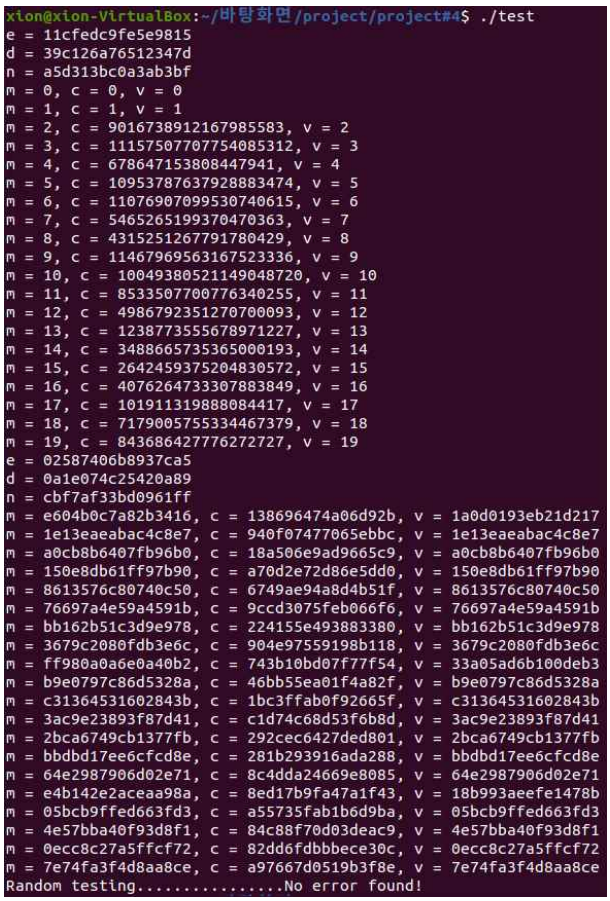
지역 함수로 선언한 mod_pow를 이용해 $M^k \bmod n$을 계산한다. 이때 $m \geq n$ 이면 m의 값이 범위를 넘었으므로 오류로 처리해서 1을 return, 그 외의 경우에는 0을 return한다.

2. compile

```
xion@xion-VirtualBox:~/바탕화면/project/project#4$ make
gcc -Wall -c test.c
test.c: In function 'main':
test.c:21:58: warning: format '%llx' expects argument of
   21 |         printf("Error: RSA key is not 64 bits: r
      |
      |
      |
      |
gcc -Wall -c mRSA.c
gcc -Wall -o test test.o mRSA.o -lbsd
```

test.c를 수정하지 않고 그대로 사용해 생기는 format warning을 제외한다면 모두 정상적으로 compile 되었다.

3. result



p, q, e, d이 모두 랜덤하게 설정되기 때문에 예상출력과는 값이 다르지만 최종적으로 Random testing에서 No error found!가 출력되는 것을 확인할 수 있다. '>' 터미널 명령어를 통해 결과를 mRSA.txt에 저장하여 별도로 제출한다.

4. feedback

맨 처음 구현한 초기 버전의 경우 약 10번의 실행 중 1~2번 꼴로 Random testing 중 generate_key에 error가 발생했고, error가 발생하지 않더라도 running time이 평균 1분 이상으로 매우 비효율적이었다. 이후 최적화를 위해 코드를 둘러보다가 miller_rabin 알고리즘에서 for loop를 제거하고 매우 효율적인 비트연산으로 대체하는 등 miller_rabin 알고리즘을 최적화하였고, 최종적으로 약 100번의 실행 중에도 한 번의 error도 발생하지 않았으며, running time도 평균 5~6초 내외로 파격적으로 감소했다.