**Cryptography Project #2**

소프트웨어학부

2018044720 석예림

1. mRSA.c 소스코드

/\*

\* Copyright 2020. Heekuck Oh, all rights reserved

\* 이 프로그램은 한양대학교 ERICA 소프트웨어학부 재학생을 위한 교육용으로 제작되었습니다.

\*/

#include <stdlib.h>

#include <stdint.h>

#include "mRSA.h"

static uint64\_t gcd(uint64\_t a, uint64\_t b)

{

uint64\_t tmp;

while(b != 0){

tmp = a % b;

a = b;

b = tmp;

}

return a;

}

static uint64\_t mod\_add(uint64\_t a, uint64\_t b, uint64\_t m)

{

a = a % m;

b = b % m;

if(a + b < b) return ((a % m) + (b % m) - m) % m;

return ((a % m) + (b % m)) % m;

}

static uint64\_t mod\_sub(uint64\_t a, uint64\_t b, uint64\_t m)

{

a = a % m;

b = b % m;

if(a < b) return ((a % m) - (b % m) + m) % m;

return ((a % m) - (b % m)) % m;

}

static 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;

}

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;

}

static 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 = 0, tmp;

while(d1 > 1){

q = d0 / d1;

tmp = mod\_sub(d0 ,mod\_mul(q, d1, m), m);

d0 = d1;

d1 = tmp;

tmp = mod\_sub(x0, mod\_mul(q, x1, m), m);

x0 = x1;

x1 = tmp;

}

if(d1 == 1) return (x1 > 0 ? x1 : x1 + m);

return 0;

}

static int miller\_rabin(uint64\_t n)

{

uint64\_t a[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

uint64\_t i, j, q, k, l, flg;

for(i = 0; i < 12; i++){

if(n == a[i]) return PRIME;

}

q = (n - 1);

k = 0;

while(k % 2 == 0){

q = q / 2;

k = k + 1;

}

for(i = 0; i < 12; i++){

flg = COMPOSITE;

if(mod\_pow(a[i], q, n) == 1) flg = PRIME;

l = 1;

for(j = 0; j < k; j++){

if(mod\_pow(mod\_pow(a[i], q, n),l, n) == n-1){

flg = PRIME;

}

l \*= 2;

}

if(flg == COMPOSITE) return COMPOSITE;

}

return flg;

}

/\*

\* mRSA\_generate\_key() - generates mini RSA keys e, d and n

\* Carmichael's totient function Lambda(n) is used.

\*/

void mRSA\_generate\_key(uint64\_t \*e, uint64\_t \*d, uint64\_t \*n)

{

uint64\_t p = 0;

uint64\_t q = 0;

uint64\_t Lambda = 0;

while (p \* q < MINIMUM\_N) // p, q 랜덤 선택

{

while (1)

{

arc4random\_buf(&p, sizeof(uint32\_t));

if (miller\_rabin(p)) break;

}

while (1)

{

arc4random\_buf(&q, sizeof(uint32\_t));

if (miller\_rabin(q)) break;

}

}

\*n = p \* q; // n 생성

Lambda = ((p - 1)\*(q - 1))/gcd(p - 1, q - 1);

while (1) // e, d 생성

{

arc4random\_buf(e,sizeof(uint64\_t));

if ((1 < \*e) && (\*e < Lambda) && (gcd(\*e,Lambda) == 1))

{

\*d = mul\_inv(\*e,Lambda);

if ((1<\*d) && (\*d<Lambda)) break;

}

}

}

/\*

\* mRSA\_cipher() - compute m^k mod n

\* If data >= n then returns 1 (error), otherwise 0 (success).

\*/

int mRSA\_cipher(uint64\_t \*m, uint64\_t k, uint64\_t n)

{

if(\*m >= n) return 1;

\*m = mod\_pow(\*m, k, n);

return 0;

}

2. 코드 내 함수 설명

* gcd, mod\_add, mod\_sub, mod\_mul, mod\_pow, mod\_inv, miller\_rabin 함수는 앞에서 한 프로그램과 프로젝트와 동일
* mRSA\_generate\_key : arc4random\_buf()를 사용해서 랜덤한 수를 선택해 주고 그 수가 prime인지 miller\_rabin()함수로 확인해 주고, p\*q>= MNIMUM\_N일때 까지 p와 q를 구한다. 이렇게 구한 p, q를 곱하여 n을 생성해준다.

Euler totient function대신 Carmichael’s totient function인 Lambda를 사용하여 키를 생성한다. 그다음 1 < e < , gcd(e, ) = 1인 랜덤한 e를 선택하고, ed 1 mod 를 만족하는 d를 구하면 암호화에 필요한 키가 생성된다.

* mRSA\_cipher : mk mod n을 계산하는데 m >= n 이면 오류처리 해주고 오류가 발생하지 않으면 0을 반환한다.

3. 실행 결과

Text

Description automatically generated