2019007329 이지현 proj4

1. 함수 설명

mod_add 함수

proj3의 mod_add 함수를 사용하였다. a+b mod m을 계산하는 함수이다. 오버플로우를 방지하기 위해 a-(m-b)를 계산한다. 만약 a+b >= m이라면, (a-(m-b)) % m을 return하고, 그렇지 않으면 (a+b) % m을 return한다. 계산을 더욱 빠르게 하기 위해 a값과 b값을 m으로 나눈 나머지를 a, b 값으로 다시 할당한 후 return 계산을 진행하였다.

mod_sub 함수

proj3의 mod_sub 함수를 사용하였다. a-b mod m을 계산하는 함수이다. a < b라면 결과가 음수가 되므로 m을 더해서 양수로 만든 값을 return한다. 만약 a < b라면 (a-b+m) % m을 return하고, 그렇지 않으면 (a-b) % m을 return한다. 계 산을 더욱 빠르게 하기 위해 a값과 b값을 m으로 나눈 나머지를 a, b값으로 다시 할당한 후 return 계산을 진행하였다

mod_mul 함수

proj3의 mod_mul 함수를 사용하였다. a*b mod m을 계산하는 함수이다. a*b에서 오버플로우를 방지하기 위해 덧셈을 사용하여 빠르게 계산할 수 있는 double addition 알고리즘을 사용하였다.

mod_pow 함수

proj4의 mod_pow 함수를 사용하였다. a^b mod m을 계산하는 함수이다. a^b에서 오버플로우를 방지하기 위해 곱셈을 사용하여 빠르게 계산할 수 있는 square multiplication 알고리즘을 사용하였다.

gcd 함수

proj1의 gcd 함수를 사용하였다. a가 0이면 b가, b가 0이면 a가 최대공약수 이므로 a == 0 || b == 0일 경우 a+b를 return해 주고 (a나 b 둘 중 하나는 무조건 0이므로 더해도 상관 없음), a나 b가 0이 아닐 경우 a는 b가 되고, b 는 a % b 되도록 temp 변수에 a 값을 저장해서 계산해 준다.

mul_inv 함수

proj1의 mul_inv를 응용하였다. proj1의 mul_inv 함수는 int type이기 때문에 uint64_t type으로 바꾸어 주었다. d0 – q * d1 계산은 mod_sub와 mod_mul 함수를 이용하여 계산하였다. (temp = mod_sub(d0 ,mod_mul(q, d1, m), m))

x0 – q * x1 계산도 마찬가지로 mod_sub(x0, mod_mul(q, x1, m), m) 와 같이 계산하였다.

만약 d1이 1이라면(a와 m이 서로소라면) x1을 return한다. 이 때, x1이 음수이면 m을 더해 양수로 변환해 준 후 return한다. d1이 1이 아닌 경우 역이 없으므로 0을 return한다.

miller_rabin 함수

proj3의 miller_rabin을 사용하였다.

계산에 들어가기 앞서 n이 0이나 1일 경우, 또는 n이 2보다 큰 짝수일 경우 소수가 아니기 때문에 if문을 통해 n 값이 앞의 조건일 경우 COMPOSITE을 return한다.

먼저 (n-1) = 2^k*q를 만족하는 k, q 값을 찾는다. q는 홀수여야 하므로 q 값이 홀수가 되면 반복문을 중단한다. n-1은 무조건 짝수이기 때문에(앞 if문 조건에 의해 n은 홀수) n-1을 홀수가 될 때까지 2로 나눈다면 그 홀수 값은 q가 되고, 몇 번 나누었는가를 count 한 값은 k가 될 것이다. 초기 q 값을 n-1로 하여 q가 홀수가 될때까지 2로 나눈 후 k를 1씩 증가시킨다.

결정적 밀러 라빈 알고리즘이므로 for문은 총 12번(리스트 a의 길이) 반복한다. 먼저 a는 1보다 크고 n-1보다 작은 값이여야 한다.

1) 만약 a[i]가 n-1보다 같거나 크면 다음 계산을 생략하기 위해 continue를 사용하였다.

mod_mul 을 이용해 (a[i]^q) mod n 값을 계산해 준 후 temp 에 할당한다.

2) 만약 temp == 1 이라면 prime 이므로 다음 계산을 생략한다.

3) 만약 find_prime(n, k, temp) == 1 이라면 find_prime 함수의 return 값이 prime(1) 이라는 의미이므로 다음 계산을 생략한다.

위의 세 단계에 모두 해당되지 않아 continue 가 실행되지 않았다면 소수가 아니므로 COMPOSITE 를 return 한다.

12 개의 a 값을 모두 검사하였을때 COMPOSITE 이 reuturn 되지 않았다면 12 번의 검사 모두 PRIME 이라는 의미이므로 for 문 도중 return 되지 않고 빠져나왔다면 PRIME 을 return 한다.

mRSA_generate_key 함수

먼저 p, q 값을 select하기 위해 arc4random_buf를 이용해 uint32_t인 랜덤 p, q를 생성한다. p, q 값이 소수인지 miller_rabin 함수를 이용해 검증하고 소수라면 break, 소수가 아니라면 다시 랜덤 q, p를 생성한다.

select한 p, q값을 이용해 n값과 lamb값을 계산한다. lamb값은 계산량을 줄여주기 위해 gcd 함수를 이용한다.

arc4random_buf를 이용해 랜덤 e 값을 생성하고, 만약 e가 gcd(e, lamb) = 1이고 1 < e < lamb 라면, break한다. e가 위의 조건에 만족하지 않는다면 다시 랜덤값을 생성한다.

select한 e값을 사용하여 ed ≡ 1 mod lamb 인 d 찾기 위해 mul_inv 함수를 이용하여 *d = mul_inv(*e,lamb) 를 계산한다.

mRSA_cipher 함수

m이 n보다 같거나 크면 1을 return하고, 1이 아니라면 m^k mod n을 계산한 후 0을 return한다.

2. 컴파일 & 실행 결과

```
lejinhy@DESKTOP-IPANOF9:/mnt/c/Users/82103/Downloads/proj
#4_1$ make all
gcc -Wall -O3 -c mRSA.c
gcc -o test test.o mRSA.o  -lbsd
```

```
lejinhy@DESKTOP-IPANOF9:/mnt/c/Users/82103/Downloads/proj
#4_1$ ./test
e = 11804453c6af7eeb
d = 1cc85fcd841f3693
n = e76a1fdc26d21389
m = 0, c = 0, v = 0
m = 1, c = 1, v = 1
m = 2, c = 9942906570348603700, v = 2
m = 3, c = 4503130837249549358, v = 3
m = 4, c = 7261506502025909653, v = 4
m = 5, c = 238043436311587995, v = 5
m = 6, c = 3538437798931751596, v = 6
m = 7, c = 6897202133628807325, v = 7
m = 8, c = 5893739382240015299, v = 8
m = 9, c = 10252227075891346713, v = 9
m = 10, c = 13903914318560195795, v = 10
m = 11, c = 13382512367231203880, v = 11
m = 12, c = 11382821460174735939, v = 12
m = 13, c = 7477012167248979883, v = 13
m = 14, c = 6869940197893089680, v = 14
m = 15, c = 4548895740321083964, v = 15
m = 16, c = 5576036340269343355, v = 16
m = 17, c = 1877450292933433046, v = 17
m = 18, c = 9308686321196201552, v = 18
m = 19, c = 5227847094947230911, v = 19
e = 00308fc2fc90df6f
d = 000aa138c3af968f
n = a5cf89724c41cac1
m = 4733540bce7fa8d6, c = 2bd8d3d01f5cb00d, v = 4733540bc
e7fa8d6
m = 19ee971e062672e4, c = 6f8672bff5374066, v = 19ee971e0
62672e4
m = c56d5dc64e475745, m may be too big
m = ef71ae2975a4f4b9, m may be too big
m = e2d0903f44faabe3, m may be too big
m = 199ac6cb02805817, c = 8eb4550fd1a7fe61, v = 199ac6cb0
2805817
m = dacac63fe5b88020, m may be too big
m = 21009af99b1a4a14, c = 5b193cd43767feb7, v = 21009af99
b1a4a14
m = 316a10abc64ee30e, c = 1ed9a7253a92499c, v = 316a10abc
64ee30e
m = 3e0ff178644dd791, c = 9e42299c16b7d077, v = 3e0ff1786
44dd791
m = a0016f13d5d3b373, c = 7fdb750751e3a2a7, v = a0016f13d5d3b373
```

```
m = dcd443433ca3b7c0, m may be too big
m = 12985ab919e0f8e7, c = 7d872a56de6c5ff4, v = 12985ab919e0f8e7
m = 6f5a4f94a8c818a7, c = 17efb344974339f3, v = 6f5a4f94a8c818a7
m = 543a4c5855067f31, c = 40ff9a8a0e6f97af, v = 543a4c5855067f31
m = 98604491aa99e83b, c = 7d4376b4aefa8d7a, v = 98604491aa99e83b
m = 42f2084d5724a2c9, c = 0fa398aa0baede73, v = 42f2084d5724a2c9
m = ee52873d4b5f3ab8, m may be too big
m = deaee421aac3d531, m may be too big
m = d342a1add3a9186e, m may be too big
Random testing.................PASSED
```

3. 설명 & 소감

mini RSA를 구현하였다. RSA 구현을 위한 지역 함수들이 이전에 이미 만들어 둔 함수들을 이용해서 사용하면 되었기 때문에 큰 어려움은 없었다. 간단한 코드로 RSA 구조를 확인할 수 있어서 RSA를 파악하는데 많은 도움이 되었다. 이전에 구현하였던 함수들을 이후 과제의 지역 함수로 사용하며 배웠던 이론을 어떤 방식으로 적용시켜야 하는지 파악할 수 있었다.

4. 문제점

눈에 띄는 문제점은 발견하지 못했다. 간단한 RSA였기 때문에 큰 어려움 없이 구현할 수 있었다.