암호학

Project #3

소프트웨어학부
2017012251
윤영훈

1. source code 설명

① mod_add

```
13  uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m)
14  {
15      a = a - (a / m) * m;
16      b = b - (b / m) * m;
17
18      return ((a >= m-b) ? a-(m-b) : a+b);
19  }
```

while (a >= m) { a = a - m; } 연산이 a가 m보다 과도하게 크다면 시간이 매우 오래걸린다는 점을 고려하여 a mod m, b mod m을 계산할 새로운 계산법을 도입하였다. '/' 연산자는 나머지를 버린 '몫'을 계산하므로 기존의 a에서 (a / m) * m 을 빼준다면 그 남은 값이 나머지 (a mod m)가 된다. b도 동일한 연산을 수행하여 최종적으로 a, b가 m보다 과도하게 큰 값이더라도 매우 빠른 시간 내에 a mod m, b mod m을 얻을 수 있게 되었다.

이후 a+b가 m보다 크다면 m을 빼주고, 그렇지 않다면 그대로 return한다. 이때 a+b에서 overflow가 유발될 수 있으므로 수식을 다음과 같이 변경하여 overflow를 예방하였다.

변경 전	변경 후
a+b >= m	a >= m-b
a+b-m	a-(m-b)

② mod_sub

```
25  uint64_t mod_sub(uint64_t a, uint64_t b, uint64_t m)
26  {
27      a = a - (a / m) * m;
28      b = b - (b / m) * m;
29
30      return ((a < b) ? a-b+m : a-b);
31  }
```

mod_add와 동일한 과정으로 a mod m, b mod m을 구한다. 이후 a-b가 음수인 경우 m을 더해주고, 그렇지 않은 경우는 그대로 return한다.

③ mod_mul & mod_pow

```
45  uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m)
46  {
47      uint64_t r = 0;
48
49      while (b > 0){
50          if (b & 1) r = mod_add(r, a, m);
51          b = b >> 1;
52          a = mod_add(a, a, m);
53      }
54
55      return r;
56  }
```

```
70  uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m)
71  {
72      uint64_t r = 1;
73
74      while (b > 0){
75          if (b & 1) r = mod_mul(r, a, m);
76          b = b >> 1;
77          a = mod_mul(a, a, m);
78      }
79
80      return r;
81  }
```

mod_mul / mod_pow 각각 double addition / square multiplication 알고리즘을 구현하여 최종적으로 얻은 r을 return한다.

④ miller_rabin

```
24  int miller_rabin(uint64_t n)
25  {
26      if (n % 2 == 0 && n != 2) return COMPOSITE;
27
28      int k = 0;
29      uint64_t q = n-1;
30
31      while ((q % 2) == 0){
32          q /= 2;
33          k++;
34      }
35
36      for (int i=0; i<ALEN && a[i] < n-1; i++){
37          uint64_t x = mod_pow(a[i], q, n);
38          int count = 0;
39
40          if (x == 1) continue;
41
42          for (int j = 0; j < k; j++){
43              if (mod_pow(x, 1 << j, n) == n-1){
44                  count++;
45                  break;
46              }
47          }
48
49          if (count == 0) return COMPOSITE;
50      }
51      return PRIME;
52  }
```

2를 제외한 모든 짝수는 소수가 아니므로 line 26을 통해 n이 2를 제외한 짝수인 경우 COMPOSITE를 return한다. n이 홀수라면 $(n-1) = 2^k q$를 만족하는 k와 q를 line 31-34를 통해 구한다. 이후 a = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}에서 $1 < a < n-1$을 만족하는 모든 a가 다음 두 조건 중 하나라도 충족한다면 PRIME, 하나의 a라도 충족하지 못한다면 COMPOSITE를 return한다.

조건 1	$a^q \bmod n = 1$
or	
조건 2	$j = 0, 1, 2, \cdots, k-1$에 대해 $(a^q)^{2^j} \bmod n = n-1$를 만족하는 j가 존재

2. compile

```
xion@xion-VirtualBox:~/바탕화면/project2$ make
gcc -Wall -c miller_rabin.c
gcc -Wall -c mod.c
gcc -Wall -o test test.o miller_rabin.o mod.o
xion@xion-VirtualBox:~/바탕화면/project2$ ./test
<덧셈> 1234 + 5678 mod 3456 = 0
<뺄셈> 1234 - 5678 mod 3456 = 2468
<곱셈> 1234 * 5678 mod 3456 = 1340
<지수> 1234 ^ 5678 mod 3456 = 1792
---
<덧셈> 3684901700 + 3904801120 mod 4294901760 = 3294801060
<뺄셈> 3684901700 - 3904801120 mod 4294901760 = 4075002340
<곱셈> 3684901700 * 3904801120 mod 4294901760 = 2417663360
<지수> 3684901700 ^ 3904801120 mod 4294901760 = 1734737920
---
<덧셈> 18446744073709551360 + 18446744073709551598 mod 18441921395520346504 = 9645356378409950
<뺄셈> 18446744073709551360 - 18446744073709551598 mod 18441921395520346504 = 18441921395520346266
<곱셈> 18446744073709551360 * 18446744073709551598 mod 18441921395520346504 = 14923616227936587640
<지수> 18446744073709551360 ^ 18446744073709551598 mod 18441921395520346504 = 6550219153064247488
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
```

warning, error 없이 정상적으로 compile 되었다.

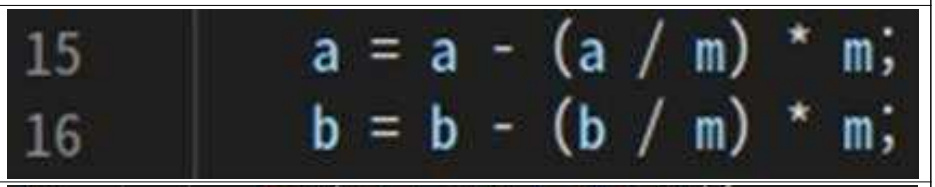
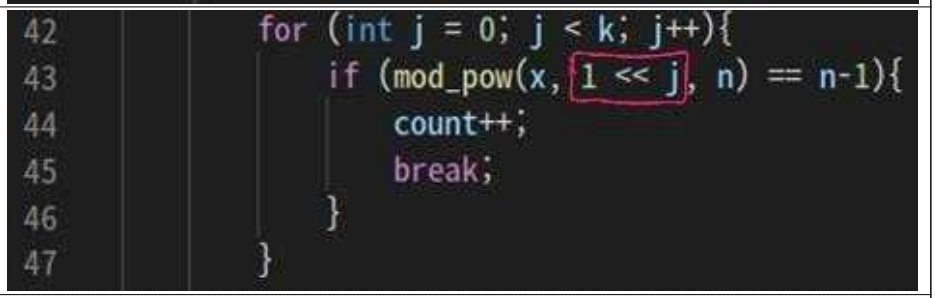
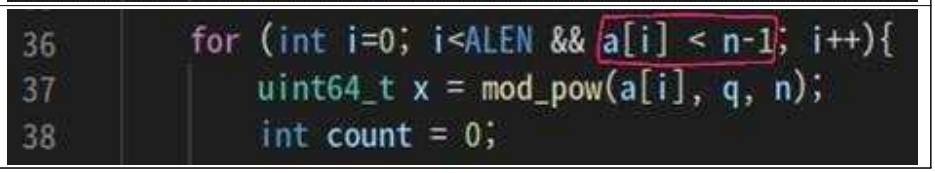
3. result

1. mod_add, mod_sub, mod_mul, mod_pow test
2. x = 2부터 10,000개의 소수 출력
3. x = 0x8000000000000000부터 100개의 소수 출력

세 가지 test를 정상적으로 수행했고 이에 대한 result 또한 예상 출력과 동일하다. '>' 터미널 명령어를 통해 결과를 miller_rabin.txt 에 저장하여 별도로 제출한다.

4. feedback

교수님께서 공지하신 범하기 쉬운 오류 3가지를 모두 개선했다고 생각한다.

범하기 쉬운 오류	개선 코드
a -= m	
pow, 2^j 계산	
1 < a < n-1	

추가적으로 miller-rabin algorithm에서 모든 a에 대해 inconclusive해야 최종적으로 PRIME 기준을 만족하는 것인데 하나의 a에 대해서만 inconclusive해도 PRIME 기준을 만족하는 것으로 잘못 이해하고 있었다. 이로 인해 구현에 어려움을 겪었고 이론부터 다시 이해하는 데 시간이 많이 소모되었다.