# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Лабораторная работа №1 по курсу «Криптография»**


Студент: Токарев Н. С.
Преподаватель: Борисов А. В.
Группа: М8О-307Б-18

Дата:
Оценка:
Подпись:

**Москва, 2021**

**Вариант №2**

Разложить каждое из чисел представленных ниже на нетривиальные сомножители.

**Первое число:**

n1 =
3523581180791504931870993551416295271017491061679972555096190205283337223522 17

**Второе число:**

n2 =
1695128485402083763773247025508607781296883851800934596605324477902989896723900984413142336870385225437965243629326745116590849908770944614057690683052539801654819522761512642822701693074249824513493644688844526263633663327921066974983001545042891090435383147221714908515772020029364695158378468844726857013205559546752704709817118834528761529676361607229919430317377276744622348039645465223497066788134123417127031908420255679798222788292548376427537395466491 59

**Выходные данные:**

Для каждого числа необходимо найти и вывести все его множетели - простые числа.

# 1 Описание

Процесс разложения числа на его простые множетели называется факторизацией. Для решения этой задачи существует множество алгоритмов, позволяющих находить множетели, используя свойства простых чисел.

Для решения задачи я выбрал алгоритм $\rho$-Полларда, как один из наиболее простых и эффективных. Данный алгоритм эффективен на небольших числах. Данный алгоритм можно назвать рандомизированным.

Для чисел, десятичнных знаков меньше 100 эффективен Метод квадратичного решета, реализация которого довольно трудоемка, поэтому я решил прибегнуть к уже готовому решению,проверенному и реализованному профессионалами - программному продукту **msieve**, который реализует в себе целый ряд алгоритмов факторизации с общим названием Общий метод решета числового поля. Этот метод считается одним из самых эффективных современных алгоритмов факторизации.

Однако 2 число имеет более 400 квадратичных знаков, факторизация которого на обычном компьютере за разумное время практически невозможна ни одним из ныне существующих алгоритмов. Однако была дана подсказка, что один из множетелей этого числа определяется к наибольший общий делитель с одним из чисел другого варанта. Поэтому необходимо найти нод числа моего варианта и числа из другого вариант. Второе же число определяется как результат деления числа моего варанта на *НОД* (по свойству делителя).

## 2 Исходный код

Реализация алгоритма $\rho$ - *Полларда* на языке C++.

```
#include <iostream>
#include <string>
#include <gmpxx.h>
#include <vector>
#include <ctime>


class po_Polard{
public:
    po_Polard(const mpz_class& num);
    po_Polard(const std::string& num);
    mpz_class get_factor();
private:
    mpz_class factor_of_num(mpz_class& num);
    void Polard_GCD(mpz_class& ans, mpz_class& x, mpz_class& y);
    void Polard_MOD(mpz_class& ans, mpz_class& x, mpz_class& y);
    void Polard_ABSOLUTE(mpz_class& ans, mpz_class& x, mpz_class& y);
    mpz_class number;
};


mpz_class po_Polard::factor_of_num(mpz_class& num){
    mpz_class x, y, ans, absolute;
    unsigned long long i = 0, stage = 2;
    x = (rand() % (number - 1)) + 1;
    y = 1;
    Polard_ABSOLUTE(absolute, x, y);
    Polard_GCD(ans, num, absolute);
    while(ans == 1){
        if(i == stage){
            y = x;
            stage <<= 1;
        }
        absolute = x * x + 1;
        Polard_MOD(x, absolute, num);
        ++i;
        Polard_ABSOLUTE(absolute, x, y);
        Polard_GCD(ans, num, absolute);
    }
    return ans;
}

mpz_class po_Polard::get_factor(){
    return factor_of_num(number);
```

```
45 }
46
47
48 po_Polard::po_Polard(const mpz_class& num){
49     srand(time(0));
50     number = num;
51 }
52
53 po_Polard::po_Polard(const std::string& str){
54     srand(time(0));
55     number = str;
56 }
57
58 void po_Polard::Polard_ABSOLUTE(mpz_class& ans, mpz_class& x, mpz_class& y){
59     x -= y;
60     mpz_abs(ans.get_mpz_t(), x.get_mpz_t());
61     x += y;
62 }
63
64
65 void po_Polard::Polard_GCD(mpz_class& ans, mpz_class& x, mpz_class& y){
66     mpz_gcd(ans.get_mpz_t(), x.get_mpz_t(), y.get_mpz_t());
67 }
68
69 void po_Polard::Polard_MOD(mpz_class& ans, mpz_class& x, mpz_class& y){
70     mpz_mod(ans.get_mpz_t(), x.get_mpz_t(), y.get_mpz_t());
71 }
72
73 using namespace std;
74
75 int main(){
76     // 95383 95393
77     std::string number = "9098870519";
78     mpz_class numberBig;
79     numberBig = number;
80     po_Polard polard(numberBig);
81     double startcl, end;
82     startcl = clock();
83     mpz_class factor = polard.get_factor();
84     std::cout << "Factor_1: " << factor << endl;
85     std::cout << "Factor_2: " << numberBig / factor << endl;
86     end = clock();
87     std::cout << "Time of working " << (end - startcl) / CLOCKS_PER_SEC << "sec" << std
           ::endl;
88     return 0;
89 }
```

# 3 Консоль

```
nikita@nikita-HP:~/MAI/Cryptography/lw1$ g++ polard.cpp -o po -lgmpxx -lgmp
// для числа 9098870519
nikita@nikita-HP:~/MAI/Cryptography/lw1$ ./po
Factor_1: 95393
Factor_2: 95383
Time of working 0.000304sec
nikita@nikita-HP:~/MAI/Cryptography/lw1$ ./msieve 3523581180791504931870993551416295271

sieving in progress (press Ctrl-C to pause)
39473 relations (19883 full + 19590 combined from 217985 partial), need 39162
sieving complete, commencing postprocessing
nikita@nikita-HP:~/MAI/Cryptography/lw1$ ls
main.cpp  msieve  msieve.dat  msieve.log  po  polard.cpp  vars.txt
nikita@nikita-HP:~/MAI/Cryptography/lw1$ cat msieve.log
Tue May 18 22:12:28 2021
Tue May 18 22:12:28 2021
Tue May 18 22:12:28 2021  Msieve v. 1.53 (SVN unknown)
Tue May 18 22:12:28 2021  random seeds: 6eba09df 5dbb0572
Tue May 18 22:12:28 2021  factoring 352358118079150493187099355141629527101749106167
Tue May 18 22:12:28 2021  no P-1/P+1/ECM available, skipping
Tue May 18 22:12:28 2021  commencing quadratic sieve (78-digit input)
Tue May 18 22:12:28 2021  using multiplier of 13
Tue May 18 22:12:28 2021  using generic 32kb sieve core
Tue May 18 22:12:28 2021  sieve interval: 12 blocks of size 32768
Tue May 18 22:12:28 2021  processing polynomials in batches of 17
Tue May 18 22:12:28 2021  using a sieve bound of 999269 (39066 primes)
Tue May 18 22:12:28 2021  using large prime bound of 99926900 (26 bits)
Tue May 18 22:12:28 2021  using trial factoring cutoff of 27 bits
Tue May 18 22:12:28 2021  polynomial 'A' values have 10 factors
Tue May 18 22:17:47 2021  39473 relations (19883 full + 19590 combined from 217985 pa
Tue May 18 22:17:47 2021  begin with 237868 relations
Tue May 18 22:17:47 2021  reduce to 56619 relations in 2 passes
Tue May 18 22:17:47 2021  attempting to read 56619 relations
Tue May 18 22:17:47 2021  recovered 56619 relations
Tue May 18 22:17:47 2021  recovered 47705 polynomials
Tue May 18 22:17:48 2021  attempting to build 39473 cycles
Tue May 18 22:17:48 2021  found 39473 cycles in 1 passes
Tue May 18 22:17:48 2021  distribution of cycle lengths:
Tue May 18 22:17:48 2021     length 1 : 19883
```

```
Tue May 18 22:17:48 2021     length 2 : 19590
Tue May 18 22:17:48 2021  largest cycle: 2 relations
Tue May 18 22:17:48 2021  matrix is 39066 x 39473 (5.7 MB) with weight 1169900 (29.64/
Tue May 18 22:17:48 2021  sparse part has weight 1169900 (29.64/col)
Tue May 18 22:17:48 2021  filtering completed in 3 passes
Tue May 18 22:17:48 2021  matrix is 28409 x 28472 (4.4 MB) with weight 923483 (32.43/c
Tue May 18 22:17:48 2021  sparse part has weight 923483 (32.43/col)
Tue May 18 22:17:48 2021  saving the first 48 matrix rows for later
Tue May 18 22:17:48 2021  matrix includes 64 packed rows
Tue May 18 22:17:48 2021  matrix is 28361 x 28472 (2.6 MB) with weight 609214 (21.40/c
Tue May 18 22:17:48 2021  sparse part has weight 392291 (13.78/col)
Tue May 18 22:17:48 2021  commencing Lanczos iteration
Tue May 18 22:17:48 2021  memory use: 2.6 MB
Tue May 18 22:17:57 2021  lanczos halted after 450 iterations (dim = 28359)
Tue May 18 22:17:57 2021  recovered 17 nontrivial dependencies
Tue May 18 22:17:57 2021  p39 factor: 562068224324090847465842314308226273321
Tue May 18 22:17:57 2021  p39 factor: 626895637985717823820028254946860326577
Tue May 18 22:17:57 2021  elapsed time 00:05:29
nikita@nikita-HP:~/MAI/Cryptography/lw1$ g++ main.cpp -o po -lgmpxx -lgmp
nikita@nikita-HP:~/MAI/Cryptography/lw1$ ./po
index: 38
factor 1:  14182411129325500872865152389301941374543994360578002782653424718793884397

    298767920444754470078418119053196659377018579455305981267274522162592010 33

    290545603198981066053369085194799881899371730623420547396714065744846128 29

    351461016162751493687123552805895463129449138772103341905913170350977470 82

factor 2: 11952329332048507049521674438309669666876305045544046285616098908190

    253377968711435212264884563859869980038070355418570031673392211221021 02

    8300427760745838691
```

# 4 Ответ

**Разложение первого числа:**

- 562068224324090847465842314308226273321
- 626895637985717823820028254946860326577

**Разложение второго числа:**

- 141824111293255008728651523893019413745439943605780027826534247187938843972987679204447544700784181190531966593770185794553059812672745221625920103329054560319898106605336908519479988189937173062342054739671406574484612829351461016162751493687123552805895463129449138772103341905913170350977470826523504806349
- 119523293320485070495216744383096696668763050455440462856160989081902533779687114352122648845638598699800380703554185700316733922112210283004277607458386 91

# 5 Выводы

В ходе данной работы я познакомился с операцией факторизации чисел. В основу криптографической системы с открытым ключом RSA положена сложность задачи факторизации произведения двух больших простых чисел. В RSA: Закрытый и открытый ключ представляет собой пару чисел. Произведение двух простых чисел случайной размерности будет являться участником пары чисел как в открытом, так и в закрытом ключе.
Таким образом с помощью факторизации, зная произведение простых чисел, можно найти данные простые числа случайной размерности. Данное решение позволит получить необходимые числа и воспроизвести открытый и закрытый ключ, однако на текущий момент при существующих алгоритмах такое вычисление крайне сложно и требует очень много вычислительных мощностей и времени.