Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Криптография»

Студент:	Д. А. Ваньков
Преподаватель:	А. В. Борисов
Группа:	М8О-307Б
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Вариант №6

Задача:

Разложить каждое из чисел представленных ниже на нетривиальные сомножители.

Первое число:

108762353292448487441247663685513658893167646930627178946128889967643172154127

Второе число:

1611765569148804856242867384258680719850010286298191204635154152942043219729044
7526886147483136114545465725205417369977940016871273001825655775233013745768986
3746546307932954424777478728351215498316173711656264574423456572770974636411400
5583231547967023025414569413122447328040416970845309432217530722433341506166879
0581352676527375610862399155982339310065668240742080964683365204046938632685331
17447729991162579236036416014409092228354404809885779998800076550137

Формат вывода:

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

1 Описание

Факторизация - это процесс разложения числа на его простые множетели. Для решения этой задачи существует множество алгоритмов, позволяющих находить множетели.

Для решения задачи я выбрал алгоритм ρ - *Полларда*, как один из наиболее простых и эффективных. Однако данный алгоритм эффективен для чисел, порядок которых много меньше порядка в задании. После 2 часов работы алгоритма, я решил, что стоит найти другой способ разложения.

С помощью метода квадратичного решета, реализованного в библиотеке **msieve**, которая помимо данного метода использует целый ряд алгоритмов для разложения. На данный момент это самый быстрый способ разложить число с большим количеством знаков. С помощью данной библиотеки удалось разложить первое число за примерно 10 секунд.

Однако этот алгоритм способен раскладывать числа порядка не более 330, а 2 число имеет порядок 400 квадратичных знаков, факторизация которого на обычном компьютере за разумное время практически невозможна ни одним из существующих алгоритмов. С помощью подсказки я заметил, что один из множетелей этого числа определяется к наибольший общий делитель с одним из чисел другого варанта. Поэтому я быстро написал программу, пребирающую все числа других варантов, определяющую их *НОД* с числом моего варанта и выводящий его, если он $\neq 1$. Второе же число определяется как результат деления числа моего варанта на *НОД*.

2 Исходный код

Реализация алгоритма ρ - *Полларда* на языке python.

```
def is_prime(N):
    if N in (0, 1):
        return False
    if N == 2:
        return True
    if N % 2 == 0:
        return False
    s = N - 1
    while s % 2 == 0:
        s //= 2
    for i in range(50):
        a = randint(1, N - 1)
        exp = s
        mod = pow(a, exp, N)
        while exp != N - 1 and mod != 1 and mod != N - 1:
            mod = mod * mod % N
            exp *= 2
        if mod != N - 1 and exp % 2 == 0:
            return False
    return True

def f(x, a, b):
    return a * x ** 2 + b

def find_all_factors(prime_factors, all_factors):
    all_factors.append(1)
    all_factors.append(prime_factors[0])
    for i in range(1, len(prime_factors)):
        tmp = []
        for f in all_factors:
            if f * prime_factors[i] not in all_factors:
                tmp.append(f * prime_factors[i])
        all_factors += tmp
    all_factors.sort()

def find_factor(n):
    maxiterssq = pi / 4 * n
    x = randint(1, n - 1)
    y = x
    d = 1
    iters = 0
    a = randint(1, n - 1)
    b = randint(1, n - 1)
    while d in (1, n):
        if iters ** 2 > maxiterssq:
            a = randint(1, n - 1)
```

```
            b = randint(1, n - 1)
            x = randint(1, n - 1)
            y = x
            iters = 0
        x = f(x, a, b) % n
        y = f(f(y, a, b), a, b) % n
        d = gcd(abs(x - y), n)
        iters += 1
    return d

def find_prime_factor(n, factors):
    if is_prime(n):
        factors.append(n)
    else:
        tmp = n // find_factor(n)
        find_prime_factor(tmp, factors)

def factor(n, factors):
    while n % 2 == 0:
        factors.append(2)
        n //= 2
    while n % 3 == 0:
        factors.append(3)
        n //= 3
    while n > 1:
        find_prime_factor(n, factors)
        n //= factors[-1]

def pollard_rho(n):
    factors = []
    factor(n, factors)
    factors.sort()
    all_factors = []
    find_all_factors(factors, all_factors)
    return all_factors
```

3 Тестирование

```
(base) chappybunny@chappybunny:~/Crypto/msieve-1.53$ ./msieve -v 1087623
53292448487441247663685513658893167646930627178946128889967643172154127
```

```
Msieve v. 1.53 (SVN unknown)
Sat Mar 14 14:51:21 2020
random seeds: ebf1cbda 21b05795
factoring 1087623532924484874412476636855136588931676469306271
78946128889967643172154127 (78 digits)
no P-1/P+1/ECM available,skipping
commencing quadratic sieve (78-digit input)
using multiplier of 23
using generic 32kb sieve core
sieve interval: 12 blocks of size 32768
processing polynomials in batches of 17
using a sieve bound of 921203 (36471 primes)
using large prime bound of 92120300 (26 bits)
using trial factoring cutoff of 26 bits
polynomial 'A'values have 10 factors
restarting with 19688 full and 187790 partial relations

36734 relations (19688 full + 17046 combined from 187790 partial),need 36567
sieving complete,commencing postprocessing
begin with 207478 relations
reduce to 51747 relations in 2 passes
attempting to read 51747 relations
recovered 51747 relations
recovered 38972 polynomials
attempting to build 36734 cycles
found 36734 cycles in 1 passes
distribution of cycle lengths:
length 1 : 19688
length 2 : 17046
largest cycle: 2 relations
matrix is 36471 x 36734 (5.3 MB) with weight 1100605 (29.96/col)
sparse part has weight 1100605 (29.96/col)
filtering completed in 4 passes
matrix is 25027 x 25091 (4.0 MB) with weight 838956 (33.44/col)
sparse part has weight 838956 (33.44/col)
```

```
saving the first 48 matrix rows for later
matrix includes 64 packed rows
matrix is 24979 x 25091 (2.4 MB) with weight 579402 (23.09/col)
sparse part has weight 374482 (14.92/col)
commencing Lanczos iteration
memory use: 2.4 MB
lanczos halted after 396 iterations (dim = 24979)
recovered 18 nontrivial dependencies
p39 factor: 260951289862485772644727258162652873363
p39 factor: 416791782672403295662841737728685758229
elapsed time 00:00:08
```

```
/home/chappybunny/PycharmProjects/NM_1_4/venv/bin/python
/home/chappybunny/PycharmProjects/NM_1_4/Cr1.py
40
First number: 163397696065821074680902655996825570159
70679523604590652155946096257851907885610572564896855
65690727114066165297231829395018127947226623668148836
31619640072792920581850719503493330646427755230896373
11981469057198581127811557725160954236258017514857831
37390808982446963816652600844796433894347926454219087
12913
Second number: 9.86406545475122e+153
```

4 Ответ

Разложение первого числа:

- p39 factor: 260951289862485772644727258162652873363
- p39 factor: 416791782672403295662841737728685758229

Разложение второго числа:

- 16339769606582107468090265599682557015970679523604590
65215594609625785190788561057256489685565690727114066
16529723182939501812794722662366814883631619640072792
92058185071950349333064642775523089637311981469057198
58112781155772516095423625801751485783137390808982446
96381665260084479643389434792645421908712913
- 9.86406545475122e+153

5 Выводы

Эта работа сама по себе не вызвала особых трудностей, если не брать в счет то, что я никогда не сталкивался с факторизацией настолько больших чисел. Только с помощью сторонней библиотеки удалось разложить только первое число. Второе же удалось разложить только прибегнув к хитрому способу.

Кроме того, оценив наколько сложно факторизовать число, количество знаков в котором превышает 400, я оценил надежность такого метода шифрования, как RSA.