Московский авиационный институт

(государственный технический университет)

**Факультет прикладной математики и физики**

Кафедра вычислительной математики и программирования

**Лабораторная работа №3**

по спецкурсу «Криптография»:  
Факторизация

|  |  |
| --- | --- |
| Выполнил: | Баскаков О.А. |
| Группа: | 08-306 |
| № по списку: | 2 |
| Преподаватель: | Рисенберг Д.В. |
| Оценка: |  |
| Дата: |  |

Москва

2011 г.

# Задание

Необходимо написать программу на языке C++, C# или Python, реализующую алгоритм факторизации. Должны поддерживаться числа длиннее 64 бит.

*Вариант №3.*

-метод Полларда.

### Исходные коды на языке Python:

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

def NOD( a, b):

"""Euclidean algorithm"""

if b == 0:

return a

else:

return NOD(b, a % b)

def my\_sqrt(n):

if (n < (1<<44)) : return int(sqrt(n))

l = (1<<22)

r = n

while( abs(l-r)>1):

t = (l+r)//2

if (t\*t>n): r = t

else: l = t

return l

#~ Простейший алгоритм разложения на множители

def factorization(n):

sqrt\_n = my\_sqrt(n)

z = []

i = 2

while(n != 1):

while (n%i == 0):

n = n // i

sqrt\_n = my\_sqrt(n)

z.append(i)

i+=1

if (i>sqrt\_n):break

if (n!=1): z.append(n)

return z

#~ Метод Полларда \*все гениальное просто\*

def Pollard(n, k = 100500):

"""Pollard's rho algorithm"""

if( Miller\_Rabin(n) ): return n

x = y = 2

for i in range(k):

x = mod(x\*x +1, n)

y = mod(y\*y +1, n)

y = mod(y\*y +1, n)

d = NOD( abs(x-y), n)

if (1 < d)and(d < n):

return Pollard(d)

return None

def is\_square(n):

t = my\_sqrt(n)

return (t\*t == n)

#~ Метод Ферма для близких делителей

def Ferma(n, itt = 100500):

"""Fermat's factorization method"""

a = my\_sqrt(n) + 1

b2 = a\*a - n

while not is\_square(b2):

b2 += 2\*a + 1 # equal b2 = a\*a - n

a += 1

itt -= 1

if (itt == 0): return None

return a - my\_sqrt(b2)

#~ Вспомогательные функции

def product(L):

mul = lambda x, y: x\*y

return reduce(mul, L)

def mod (a, b):

return a % b

### Тестирование:

oleg@debian:~/crypto$ ./3.py help

This programm use Pollard's pho algorithm

for integer factorization.

-help for this text

-primes get some primes number

-primes lim some primes number more than lim

num\_1\*num\_2\*...\*num\_k - factorization of number

none - standart test %

oleg@debian:~/crypto$ time ./3.py

factorization of 9372086472077162188799696296204855972388191052081362406893026506475796964921576092048327809

multiplier 3

multiplier 11

multiplier 59

multiplier 109

multiplier 997

multiplier 8689

multiplier 1061323

multiplier 1903619

multiplier 1088093

multiplier 1059259

multiplier 1059323

multiplier 45300907

multiplier 337293247297

multiplier 8238276495697

multiplier 16417377581384821

Check: 2048327809 == 2048327809 True

real 1m36.457s

user 1m35.918s

sys 0m0.020s

Отметим, что нашелся множитель 8238276495697 порядка 2^43.

oleg@debian:~/crypto$ ./3.py -primes

1971062777

171086189

2148793769

971072423

1971062777

1971060239

971072423

oleg@debian:~/crypto$ ./3.py -primes 30000000000001000500

30000000000001000583

oleg@debian:~/crypto$ ./3.py -primes 30000000000009000500

30000000000009000529

30000000000009000569

Получили простые числа для тестирования.

oleg@debian:~/crypto$ ./3.py 30000000000009000529\*30000000000001000583

factorization of 900000000000300033360000009005776308407

Fail, try to Ferma

multiplier 30000000000001000583

multiplier 30000000000009000529

Check: 5776308407 == 5776308407 True

Отметим, что не просто было придумать ситуацию, в которой Поллард показал худший результат, чем метод Ферма для близких множителей.

oleg@debian:~/crypto$ ./3.py 123456789012345678900000000047

factorization of 123456789012345678900000000047

multiplier 97

multiplier 733

multiplier 69325762691

multiplier 25046359628617

Check: 47 == 47 True

oleg@debian:~/crypto$ time ./3.py 9999999999999999000000000047

factorization of 9999999999999999000000000047

multiplier 119983

multiplier 169652033

multiplier 491271098187073

Check: 47 == 47 True

real 0m0.236s

user 0m0.212s

sys 0m0.004s

На сложных примерах метод Полларда очень эффективен с малыми сомножителями.

# Выводы

В отличие от задачи распознавания простоты числа, факторизация является сложной задачей. Предполагаемая сложность задачи факторизации лежит в основе стойкости некоторых алгоритмов шифрования с открытым ключом, таких как RSA.

-метод Полларда факторизации чисел достаточно простой и эффективный. Несмотря на свою эвристичность на практике он применяется широко. Интересно происхождение названия метода. Поскольку кольцо вычетов модулю конечно и, каждое значение последовательности зависит от предыдущего, эта последовательность начнет повторяться. Т.е. визуально она похожа на греческую букву .