Лабораторная работа № 1 по курсу криптографии

Выполнила студентка группы М8О-307Б *Довженко Анастасия.*

Условие

Разложить каждое из чисел n1 и n2 на нетривиальные сомножители.
Вариант 7.
n1=10876235329244848744124766368551365889316764693062717894612888996764317215412 7
n2=16117655691488048562428673842586807198500102862981912046351541529420432197290 4
47526886147483136114545465725205417369977940016871273001825655775233013745768986374
65463079329544247774787283512154983161737116562645744234565727709746364114005583231
54796702302541456941312244732804041697084530943221753072243334150616687905813526765
27375610862399155982339310065668240742080964683365204046938632685331174477299911625
79236036416014409092228354404809885779998800076550137

Метод решения

Был использован Ро-алгоритм Полларда. Изначально ищем все простые делители исходного числа, затем, как несложно догадаться получаем все существующие делители перемножением простых делителей. Как ищутся простые делители? Случайным образом выбирается число x, на каждой итерации вычисляется значение функции $f^i(x)$. В качестве функции взята $f(x) = ax^2 + b$ для случайных a и b. На i-ом шаге получаем значения $x_i = f^i(x_0)(modn), y_i = x_{2i} = f^{2i}(x_0)(modn)$. $gcd(abs(x-y), n)$ даст нетривиальный сомножитель n. Функция выбирается каждый раз, когда количество итераций превышает $O(\sqrt{n})$. Проверка простоты числа осуществляется с помощью теста Миллера-Рабина, который позволяет выполнять проверку быстрее.

Полученная реализация, проработав ночь, все еще искала сомножители первого числа, поэтому было решено воспользоваться готовой реализацией метода решета числового поля – msieve По совету старших коллег при факторизации второго числа был использован хак: первый множитель находится как НОД с числом другого варианта, а второй множитель – делением.

Результат работы программы

```
karma@mydruzhok:~/Downloads/msieve-1.53$ ./msieve 108762353292448487 44
124766368551365889316764693062717894612888996764317215412 7

sieving in progress (press Ctrl-C to pause)
36960 relations (19767 full + 17193 combined from 190013 partial), need
36567
sieving complete, commencing postprocessing
```

```
karma@mydruzhok:~/Downloads/msieve-1.53$ cat msieve.log
Sat Mar  9 21:38:59 2019
Sat Mar  9 21:39:00 2019
Sat Mar  9 21:39:00 2019  Msieve v. 1.53 (SVN unknown)
Sat Mar  9 21:39:00 2019  random seeds: d7a92415 3ab6e5a2
Sat Mar  9 21:39:00 2019  factoring 1087623532924484874412476636855136
58893167646930627178946128889967643172154127 (78 digits)
Sat Mar  9 21:39:00 2019  no P-1/P+1/ECM available, skipping
Sat Mar  9 21:39:00 2019  commencing quadratic sieve (78-digit input)
Sat Mar  9 21:39:00 2019  using multiplier of 23
Sat Mar  9 21:39:00 2019  using generic 32kb sieve core
Sat Mar  9 21:39:00 2019  sieve interval: 12 blocks of size 32768
Sat Mar  9 21:39:00 2019  processing polynomials in batches of 17
Sat Mar  9 21:39:00 2019  using a sieve bound of 921203 (36471 primes)
Sat Mar  9 21:39:00 2019  using large prime bound of 92120300 (26 bits)
Sat Mar  9 21:39:00 2019  using trial factoring cutoff of 26 bits
Sat Mar  9 21:39:00 2019  polynomial 'A' values have 10 factors
Sat Mar  9 21:40:56 2019  36960 relations (19767 full + 17193 combined
from 190013 partial), need 36567
Sat Mar  9 21:40:56 2019  begin with 209780 relations
Sat Mar  9 21:40:56 2019  reduce to 51987 relations in 2 passes
Sat Mar  9 21:40:56 2019  attempting to read 51987 relations
Sat Mar  9 21:40:57 2019  recovered 51987 relations
Sat Mar  9 21:40:57 2019  recovered 39223 polynomials
Sat Mar  9 21:40:57 2019  attempting to build 36960 cycles
Sat Mar  9 21:40:57 2019  found 36960 cycles in 1 passes
Sat Mar  9 21:40:57 2019  distribution of cycle lengths:
Sat Mar  9 21:40:57 2019     length 1 : 19767
Sat Mar  9 21:40:57 2019     length 2 : 17193
Sat Mar  9 21:40:57 2019  largest cycle: 2 relations
Sat Mar  9 21:40:57 2019  matrix is 36471 x 36960 (5.4 MB) with weight
1107501 (29.96/col)
Sat Mar  9 21:40:57 2019  sparse part has weight 1107501 (29.96/col)
Sat Mar  9 21:40:57 2019  filtering completed in 3 passes
Sat Mar  9 21:40:57 2019  matrix is 25132 x 25196 (4.0 MB) with weight
838109 (33.26/col)
Sat Mar  9 21:40:57 2019  sparse part has weight 838109 (33.26/col)
Sat Mar  9 21:40:57 2019  saving the first 48 matrix rows for later
Sat Mar  9 21:40:57 2019  matrix includes 64 packed rows
Sat Mar  9 21:40:57 2019  matrix is 25084 x 25196 (2.4 MB) with weight
575627 (22.85/col)
Sat Mar  9 21:40:57 2019  sparse part has weight 373813 (14.84/col)
```

```
Sat Mar  9 21:40:57 2019  commencing Lanczos iteration
Sat Mar  9 21:40:57 2019  memory use: 2.4 MB
Sat Mar  9 21:41:02 2019  lanczos halted after 398 iterations (dim =
25074)
Sat Mar  9 21:41:02 2019  recovered 13 nontrivial dependencies
Sat Mar  9 21:41:02 2019  p39 factor: 260951289862485772644727258162652873363
Sat Mar  9 21:41:02 2019  p39 factor: 416791782672403295662841737728685758229
Sat Mar  9 21:41:02 2019  elapsed time 00:02:02
karma@mydruzhok:~/Downloads/msieve-1.53$ cd ~/mai_study/Crypto/lab1/
karma@mydruzhok:~/mai_study/Crypto/lab1$ python main.py
Original number: 1611765569148804856242867384258680719850010286298191204635154152942043219729044752688614748313611454546572520541736997794001687127300182565577523301374576898637465463079329544247774787283512154983161737116562645744234565727709746364114005583231547967023025414569413122447328040416970845309432217530722433341506166879058135267652737561086239915598233931006566824074208096468336520404693863268533117447729991162579236036416014409092228354404809885779998800076550137
Factors:
163397696065821074680902655996825570159706795236045906521559460962578519078856105725648968556569072711406616529723182939501812794722662366814883631619640072792920581850719503493330646427755230896373119814690571985811278115577251609542362580175148578313739080898244696381665260084479643389434792645421908712913
9.86406545475122e+153
Time:  0:00:00.000077
```

Выводы

Факторизация целых чисел обеспечивается основной теоремой арифметики. По основной теореме арифметики каждое натуральное число имеет единственное разложение на простые множители. Существует множество алгоритмов факторизации целого, с помощью которых можно факторизовать любое натуральное число до состава его простых множителей с помощью рекуррентных формул. Однако, для очень больших чисел эффективный алгоритм пока неизвестен.

Факторизация больших чисел является задачей большой сложности. Не существует никакого известного способа, чтобы решить эту задачу быстро. Её сложность лежит в основе некоторых алгоритмов шифрования с открытым ключом, таких как RSA. Множество областей математики и информатики находят применение в решении этой задачи. Среди них: эллиптические кривые, алгебраическая теория чисел и квантовые вычисления.

Листинг программного кода

```python
from random import randint
from math import gcd, pi
from functools import reduce
from datetime import datetime

CHECK_NUM = 15987565442108608120026832525046666312840385351549793409109648246739235786392263979181344291927370058541881779770591778582438559908039812756656909129755340910413617018434655781017338634797816807916559595783204421083716340483743135242021931986948945364524716468688251447430144529579127439202399544735343744226477480201653067693793961900445995131103930624613028392443567547410653207750115147747231558637315951828928227907098432963750752726519026414605041032917753 61

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


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 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]
```

```python
def find_all_factors(prime_factors, all_factors):
    if len(prime_factors) == 0:
        all_factors.append(1)
    elif len(all_factors) == 0:
        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 get_info(num, factors, start_time):
    print("Original number: {0}".format(num))
    print("Factors:")
    print(*factors, sep='\n')
    print("Time: {0}".format(datetime.now() - start_time))


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


if __name__ == '__main__':
    n = 0
    with open('test1', 'r') as file:
        n = int(file.read())
    start_time = datetime.now()
    factors = pollard_rho(n)
    factors = factors[1:len(factors) - 2]
    get_info(n, factors, start_time)

    factors.clear()
    with open('test2', 'r') as file:
```

```
    n = int(file.read())
start_time = datetime.now()
factors.append(gcd(n, CHECK_NUM))
factors.append(n / factors[0])
get_info(n, factors, start_time)
```