**Лабораторная работа №1 по теме «Алгоритм RSA»**

Выполнил студент гр.Б8117-02.03.01

Михайлов Денис

**Цель:**

Реализовать алгоритм RSA, а также тесты на простоту числа Миллера-Рабина и Соловея-Штрасса. Тесты на простоту следует проверить по времени.

**Теория:**

**Алгоритм RSA:**

RSA (Rivest, Shamir, Ableman – фамилии создателей) – это криптографический алгоритм с открытым ключом, который основывается на вычислительной сложности задачи факторизации больших целых чисел.

**Алгоритм:**

1. Выбираются два различных случайных простых числа и заданного размера, например 1024 бита каждое.
2. Вычисляется их произведение , которое называется *модулем*.
3. Вычисляется значение функции Эйлера от числа :
4. Выбирается целое число , взаимно простое со значением функции .
   * число называется *открытой экспонентой*;
   * время, необходимое для шифрования с использованием быстрого возведения в степень, пропорционально числу единичных бит в ;
   * слишком малые значения потенциально могут ослабить безопасность схемы RSA.
5. Вычисляется число , называемое секретной экспонентой, которая удовлетворяет сравнению:
6. Пара называется открытым ключом RSA;
7. Пара называется закрытым ключом RSA и хранится только.

*Взаимно простыми числами* называются такие числа, наибольший общий делитель которых равен .

Для ускорения работы алгоритма, используют *алгоритм быстрого возведения в степень*. Суть этого алгоритма в том, что он позволяет не работать со слишком большими числами и производить умножений вместо . К примеру, нам необходимо получить остаток от деления на . Мы можем честно возвести в степень и получить остаток от деления на . Но, чтобы облегчить задачу компьютера, мы можем брать остаток от деления на после каждого возведения в степень.

*Расширенный алгоритм Евклида* находит помимо НОД также коэффициенты и такие, что:

То есть он находит коэффициенты, с помощью которых НОД двух чисел выражается через сами эти числа.

Внести вычисление этих коэффициентов в алгоритм Евклида несложно, достаточно вывести формулы, по которым они меняются при переходе от пары к паре .

Итак, пусть мы нашли решение задачи для новой пары

и хотим получить решение для нашей пары

Для этого преобразуем величину

где – целая часть от деления на .

Подставим это в приведённое выше выражение с и и получим:

и, выполняя перегруппировку слагаемых, получаем:

Для облегчения поиска простых чисел использую разные алгоритмы. В рамках данной лабораторной работы мы рассмотрим два из них:

* тест Миллера-Рабина;
* тест Соловея-Штрассена.

**Тест Миллера-Рабина:**

Алгоритм Миллера-Рабина позволяет выполнять проверку за малое время и давать при этом достаточно малую вероятность того, что число на самом деле является составным.

Данный алгоритм опирается на проверку ряда равенств, которые выполняются для простых чисел. Если хотя бы одно такое равенство не выполняется, то число является составным.

Пусть – простое число и , где  - нечётно. Тогда для любого выполняется хотя бы одно из следующих условий:

* ;
* существует целое число такое, что .

Если проходит хотя бы одно из этих условий, то оно называется свидетелем простоты проверяемого числа.

Алгоритм параметризируется количеством раундов , от которых зависит шанс того, что проверяемое число является простым. Из теоремы Рабина следует, что если случайно выбранных чисел оказались свидетелями простоты проверяемого числа, то вероятность того, что проверяемое число является составным не превосходит .

**Тест Соловея-Штрассена:**

Этот алгоритм по свойствам похож на предыдущий, то есть он так же является вероятностным и шанс простоты числа зависит от раундов.

В данном тесте проводятся следующая проверка для :

где – символ Якоби.

Для вычисления символа Якоби необходимо воспользоваться квадратичным законом взаимности. Благодаря ему алгоритм похож на алгоритм Евклида нахождения наибольшего общего делителя двух чисел, в котором тоже аргументы на каждом шаге меняются местами. Аналогично алгоритму Евклида, при перестановке аргументов больший заменяется на остаток от деления на меньший. Это возможно благодаря периодичности символа Якоби. Однако, поскольку символ Якоби определён только при условии нечётности второго аргументы, то до перестановки выделяется чётная часть первого аргумента.

**Шифрование с помощью RSA:**

После нахождения открытого и закрытого ключей, происходит шифрование сообщения по следующему алгоритму:

* если сообщение текстовое, то его символы переводятся в числовой формат (прим. с использованием таблицы ascii);
* каждая часть зашифрованного сообщения находится по формуле

где и – части открытого ключа.

После того, как сообщение было зашифровано, оно передаётся в зашифрованном виде тому, у кого есть закрытый ключ.

**Расшифровка с помощью RSA:**

После получения обладателем закрытого ключа сообщения , происходит его расшифровка в исходное сообщение по следующей формуле:

где и – части закрытого ключа.

**Результаты работы:**

**RSA:**

{65537, 9294619} {3918241, 9294619}

[790984, 7666265, 3342612, 3342612, 1969007, 393967, 6088055, 1969007, 8128263, 3342612, 1515466, 393967, 1487280, 2004706, 1192022, 393967, 630333]

['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', ' ', '1', '2', '3', ' ', '!']

В первой строке содержится два массива значений: Свободный ключ и Секретный ключ соответственно.

Во второй строке представлено зашифрованное сообщение.

В третей строке показан результат расшифровки. Так как у нас сообщение было «***hello world 123 !***».

**Сравнения тестов на простоту:**

['0.0000185000', '0.0000054000', '0.0000044000', '0.0000039000', '0.0000049000', '0.0000042000']

['0.0003821000', '0.0000228000', '0.0000030000', '0.0000021000', '0.0000033000', '0.0000021000']

В первой строке указано время тестов метода Миллера-Рабина.

Во второй строке указано время тестов метода Соловея-Штрасса.

**Код программы RSA.**

def RSA(p, q, message):

    n = p \* q                  # Generated free and secret keys

    phi = (p - 1) \* (q - 1)

    e = 65537

    d = extended\_gcd(e, phi)[1]

    if d < 0: d += phi

    Fkey = {e, n}            # Free key

    Skey = {d, n}              # Secret key

    print(Fkey, Skey)

    m = [(ord(elem)\*\*e % n) for elem in message]  # Encryption of the message

    print(m)

    return [(chr(Exponentiation(elem, d, n))) for elem in m]   # Decrypting a message

def extended\_gcd(a, b):

    if a == 0: return b, 0, 1

    rezult, q1, q2 = extended\_gcd(b % a, a)

    t = q2 - (b // a) \* q1

    s = q1

    return rezult, t, s

def Exponentiation(elem, degree, modulus):

    c = 1

    for \_ in range(degree):

        c = (c \* elem) % modulus

    return c

p, q = 3347, 2777

message = "hello world 123 !"

m = RSA(p, q, message)

print(m)

**Код программы «Сравнение тестов простоты»**

from random import randint

from timeit import default\_timer

def MR(numb, k=30):

    d, s = factor(numb-1)

    if numb <= 1:

        return False

    if numb <= 3:

        return True

    if numb % 2 == 0:

        return False

    for \_ in range(k):

        a = randint(2, numb-2 )

        x = Exponentiation(a, d, numb)

        if not (x == 1 or x == numb - 1):

            for j in range(s-1):

                x = Exponentiation(a, 2\*\*(j+1) \* d, numb)

                if x == 1: return False

                if x == numb-1: break

        else: return True

    return True

def SS(n, k=30):

    if n % 2 == 0:

        return False

    if n % 3 == 0:

        return False

    for \_ in range(k):

        a = randint(3, n-2)

        if gcd(a, n) != 1:

            return False

        if Exponentiation(a, (n-1)//2, n) != (jacobi(a, n) % n):

            return False

    return True

def gcd(a, b):

    while b:

        a %= b

        a, b = b, a

    return a

def jacobi(a, n):

    t = 1

    while a != 0:

        while a % 2 == 0:

            a /= 2

            r = n % 8

            if r == 3 or r == 5:

                t = -t

        a, n = n, a

        if a % 4 == n % 4 == 3:

            t = -t

        a %= n

    if n == 1:

        return t

    else:

        return 0

def Exponentiation(elem, degree, modulus):

    c = 1

    for \_ in range(degree):

        c = (c \* elem) % modulus

    return c

def factor(numb):

    n = numb

    count = 0

    while(True):

        if n % 2 == 1: return int(n), count

        n /= 2

        count += 1

def test(\_range, algorithm):

    times = []

    for i in \_range:

        t = default\_timer()

        algorithm(2\*\*i)

        times.append('{:.10f}'.format(default\_timer() - t))

    return times

\_range = range(5, 11)

print(test(\_range, MR))

print(test(\_range, SS))