МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ  НЭТИ

Кафедра прикладной математики

Лабораторная работа № 2

по дисциплине «Основы криптографии»

Место для ввода текста.

Группа	ПМ-93, ПМ-92
Бригада	29
	ИВАНОВ ВЛАДИСЛАВ
	ОБЕРШТ ЕЛЕНА
Преподаватели	СТУПАКОВ И.М.
Дата	01.10.2021

Новосибирск

Задание

Написать программы, реализующие алгоритм RSA.

Генерация ключей
Прочитать из консоли числа *p* и *q*.
Вычислить *n* и *λ(n)*, где *λ* функция Кармайкла.
Сгенерировать открытую экспоненту *e (1 < e < λ(n))* и проверить что она является взаимно простой с *λ(n)*.
Вычислить закрытую экспоненту *d*, вывести открытый ключ *(n, e)* и закрытый ключ *(n, d)*.
Шифрование текста
Прочитать из консоли открытый ключ *(n, e)*.
Прочитать сообщение *M* (число меньшее *n*), зашифровать его алгоритмом RSA и вывести результат.
Расшифровка текста
Прочитать из консоли закрытый ключ *(n, d)*.
Прочитать зашифрованное сообщение, расшифровать его алгоритмом RSA и вывести результат.
(дополнительно, +2 балла) Сделать возможность шифровать текстовые сообщения произвольной длины
Для возведения в степень должен использоваться алгоритм быстрого возведения в степень. Программа должна корректно работать для значений $p < 2^{32}$.

```
import random
import re

# generates e, d, N
def generateKeys(keysize):
    p = generatePrime(keysize)
    q = generatePrime(keysize)
    N = p*q # used as the modulus for both the public and private keys
    lambdaN = lcm(p-1, q-1) # carmichael's function

    # generating e and checking whether e is coprime with lambdaN
    while True:
        e = random.randrange(pow(2, keysize-1), pow(2, keysize)) # 1 < e <= lambdaN
        if (isCoprime(e, lambdaN)):
            break
    # the built-in pow() function performs binary exponentiation, which is an
effective method of exponentiation

    # d is the modular inverse of e with respect to lambdaN
    d = modularInv(e, lambdaN)

    print('p:', p)
    print('q:', q)
    return e, d, N


# returns random prime number of size specified in keysize
def generatePrime(keysize):
    while True:
        number = random.randrange(pow(2, keysize-1), pow(2, keysize))
        if (isPrime(number)):
            return number

# computes greatest common divisor (euclidean algorithm)
def gcd(n, m):
    while m: # stops when m == 0
        n, m = m, n % m
    return n
```

```
# computes least common multiple
def lcm(n, m):
    return abs(n*m) // gcd(n, m)


# returns true if the number is prime
def isPrime(number):
    if number == 0 or number == 1: # 0 and 1 aren't prime
        return False

    # prime numbers up to 997
    low_primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149,
151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239,
241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347,
349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563,
569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659,
661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773,
787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887,
907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
    if number in low_primes: # true if in the list
        return True
    for prime in low_primes:
        if number % prime == 0: # false if divided by low prime
            return False


# perform miller-rabin primality test
    c = number - 1
    while c % 2 == 0:
        c /= 2
    for i in range(256): # prove that it's not a prime 256 times
        if not millerRabin(number, c):
            return False

    return True


# returns true if n and m are coprime (gcd is 1)
def isCoprime(n, m):
    return gcd(n, m) == 1


# modular multiplicative inverse
def modularInv(n, m):
    return pow(n, -1, m)


# performs miller-rabin primality test
def millerRabin(n, d):
    a = random.randint(2, (n-2)-2)
    x = pow(a, int(d), n) # a^d % n
    if x == 1 or x == n - 1:
        return True
# square x
    while d != n - 1:
        x = pow(x, 2, n)
        d *= 2

        if x == 1:
            return False
        elif x == n - 1:
            return True

    return False
```

```python
# encrypts the number using public key
def encryptNumber(e, N, number):
    return pow(number, e, N)


# decrypts the number using private key
def decryptNumber(e, N, number):
    return pow(number, d, N)


# encrypts each character in the text using public key
def encryptText(e, N, text):
    cipher = ''
    for c in text:
        m = ord(c) # returns an integer representing the unicode character
        cipher += str(pow(m, e, N)) + ' '

    return cipher


# decrypts the text using private key
def decryptText(d, N, cipher):
    text = ''
    parts = cipher.split()
    for part in parts:
        if part:
            c = int(part)
            text += chr(pow(c, d, N))

    return text


# reads and preprocesses large texts
def readText(path):
    text = open(path, 'r').read().splitlines()
    text = ' '.join(text)
    text = re.sub(r'[^а-яё ЁА-Я.!?:;,''""]+', '', text)
    return text


keysize = 32
e, d, N = generateKeys(keysize)

M1 = 52387
M2 = readText('data/war_and_peace_book_one.txt')

print('\nPublic key:  ('+str(N)+', '+str(e)+')')
print('Private key: ('+str(N)+', '+str(d)+')')
```

Вывод на консоль:

```
p: 2515452529
q: 2419787401

Public key:  (6086860337487787129, 4251484931)
Private key: (6086860337487787129, 35457597135777371)
```

```python
enc1 = encryptNumber(e, N, M1)
dec1 = decryptNumber(d, N, enc1)

print('Message:\n', M1, '\n')
print('Encrypted message:\n', enc1, '\n')
print('Decrypted message:\n', dec1)
```

Вывод на консоль:

```
Message:
 52387

Encrypted message:
 1324232619750050321

Decrypted message:
 52387
```

```
%%time

enc2 = encryptText(e, N, M2)
dec2 = decryptText(d, N, enc2)

print('Message lenght:', len(M2), 'symbols\n')
print('Message:\n', M2[:750]+'...\n')
print('Encrypted message:\n', enc2[:750]+'...\n')
print('Decrypted message:\n', dec2[:750]+'...\n')
```

Вывод на консоль:

```
Message lenght: 1873323 symbols

Message:
  Ну что, князь, Генуя и Лукка стали не больше как поместья, поместья фамилии
Буонапарте. Нет, я вам вперед говорю, если вы мне не скажете, что у нас война, если
вы позволите себе защищать все гадости, все ужасы этого антихриста право, я верю,
что он антихрист,  я вас больше не знаю, вы уже не друг мой, вы уже не мой верный
раб, как вы говорите. Ну, здравствуйте, здравствуйте. Я вижу, что я вас пугаю,
садитесь и рассказывайте.    Так говорила в июле  года известная Анна Павловна
Шерер, фрейлина и приближенная императрицы Марии Федоровны, встречая важного и
чиновного князя Василия, первым приехавшего на ее вечер. Анна Павловна кашляла
несколько дней, у нее был грипп, как она говорила грипп был тогда новое слово,
употреблявшееся только редкими...

Encrypted message:
 5481731813099389441 1989752114129135380 4396484786368027975 5481731813099389441
3575639428083569238 1060764110118597959 5737593229064762644 5139058721192755285
5481731813099389441 1191631474874318546 2423295179551096745 1666416152046675670
4051550301907935541 2004469841681099223 5139058721192755285 5481731813099389441
4219329329664906002 343612185099672891 2423295179551096745 4396484786368027975
1666416152046675670 5481731813099389441 578783908439827411 5481731813099389441
2346895687920828825 4396484786368027975 1191631474874318546 1191631474874318546
6036271721890170549 5481731813099389441 446020138272895328 1060764110118597959
6036271721890170549 2485016875854572729 578783908439827411 5481731813099389441
2423295179551096745 343612185099672...

Decrypted message:
  Ну что, князь, Генуя и Лукка стали не больше как поместья, поместья фамилии
Буонапарте. Нет, я вам вперед говорю, если вы мне не скажете, что у нас война, если
вы позволите себе защищать все гадости, все ужасы этого антихриста право, я верю,
что он антихрист,  я вас больше не знаю, вы уже не друг мой, вы уже не мой верный
раб, как вы говорите. Ну, здравствуйте, здравствуйте. Я вижу, что я вас пугаю,
садитесь и рассказывайте.    Так говорила в июле  года известная Анна Павловна
Шерер, фрейлина и приближенная императрицы Марии Федоровны, встречая важного и
чиновного князя Василия, первым приехавшего на ее вечер. Анна Павловна кашляла
несколько дней, у нее был грипп, как она говорила грипп был тогда новое слово,
употреблявшееся только редкими...

CPU times: user 31.6 s, sys: 115 ms, total: 31.7 s
Wall time: 32.5 s
```

Links

- RSA - Wikipedia
- Miller–Rabin primality test
- Coprime integers
- Modular multiplicative inverse
- Euclidean algorithm
- Carmichael function
- Modular arithmetic
- Exponentiation by squaring
- Implementation of the built-in pow() function