Автор: Рудченко Д.О.

Група: КН-М922А

Лабораторна робота №3

Асиметричне шифрування. Алгоритм RSA

Мета: Дослідити і реалізувати механізм асиметричного алгоритму шифрування RSA.

Завдання: Розробити додаток обміну таємними посиланнями між двома клієнтами за допомогою алгоритму шифрування RSA. Реалізувати алгоритм генерації ключів (public / private keys) для алгоритму RSA. Створити ключі заданої довжини (напр. 1024 біт)

- Реалізувати та продемонструвати роботу алгоритму шифрування та дешифрування повідомлення RSA
- Підтвердити роботу реалізованого алгоритму шляхом порівняння результату кодування з існуючим алгоритмом (наприклад, використовуючи утиліту openssl або вбудовані системи шифрування обраної мови програмування).

Хід роботи:

Код програми:

```
from math import gcd
import random
from random import randint
import sys

def encodeMessage(msg):
    encodedMsg = 0

    for char in msg:
        encodedMsg = encodedMsg << 8
        encodedMsg = encodedMsg ^ ord(char)
    return encodedMsg

def getRandomPrime(primeSize):
    x = randint(1 << (primeSize - 1), (1 << primeSize) - 1)
    while not (isPrime(x)):
        x = randint(1 << (primeSize - 1), (1 << primeSize) - 1)
    return x


def isPrime(n):
    if n % 2 == 0:
        return False

    for i in range(1, 40):
```

```
        a = random.randint(1, n - 1)
        if isComposite(a, n):
            return False
    return True


def isComposite(a, n):
    t, d = decompose(n - 1)
    x = pow(a, d, n)

    if x == 1 or x == n - 1:
        return False

    for i in range(1, t):
        x0 = x;
        x = pow(x0, 2, n)
        if x == 1 and x0 != 1 and x0 != n - 1:
            return True
    if x != 1:
        return True

    return False


def decompose(n):
    i = 0
    while n & (1 << i) == 0:
        i += 1
    return i, n >> i


def getKeys(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    for i in range(2, phi):
        if gcd(phi, i) == 1:
            e = i
            break

    d = multiplicativeInverse(e, phi)

    return n, e, d


def multiplicativeInverse(e, phi):
    return extendedEuclid(e, phi)[1] % phi


def extendedEuclid(a, b):
    if b == 0:
        return a, 1, 0
    else:
        d2, x2, y2 = extendedEuclid(b, a % b)
        d, x, y = d2, y2, x2 - (a // b) * y2
        return d, x, y


try:
    modulusSize = int(sys.argv[1])

except:
    modulusSize = 1024

msg = "Danylo Rudchenko"

primeSize = modulusSize // 2
p = getRandomPrime(primeSize)
```

```
print("n = ", p)
q = getRandomPrime(primeSize)
while p == q:
    q = getRandomPrime(primeSize)

n, e, d = getKeys(p, q)

encodedMsg = encodeMessage(msg)
encryptedMsg = pow(encodedMsg, e, n)
decryptedMsg = pow(encryptedMsg, d, n)
#d, x, y = d2, y2, x2 - (a // b) * y2
#print(d, x, y)
#d = d2 - (a // b) * y2
#print(1 << (4 - 1))
print("Public key (e, n):")
print("\te = ", e)
print("\tn = ", n)
print("\nPrivate key (d, n):")
print("\td = ", d)
print("\tn = ", n)
print("\nOriginal message string:\n\t", msg)
print("\nInteger encoded message:\n\t", encodedMsg)
print("\nEncrypted message( C(M) = M^e % n ):\n\t", encryptedMsg)
print("\nDecrypted message( M(C) = C^d % n ):\n\t", decryptedMsg)
if encodedMsg == decryptedMsg:
    print("\nThe decrypted message and the original encoded message match.")
```

```
input
n =  772029293851156395727541259998428751262456044203774644732971700370104182558822094607834828692236674189937296120099962419794679807279
565125038142954596012 3
Public key (e, n):
        e =  11
        n =  895410500789131591247526210608434064230435073148622008002020896157420207348676330194141710082894382973272467815258114647634
573148188813444435033519088264305269981539193091179788226456525968195638869981575338004436464356112945854187431758183545775809092988954
828066099460837023613853784099807700 49820739613

Private key (d, n):
        d =  814009546171937810225023827825849149300395521044201825456382632870382006680614845631037918257176711793884061650234649679667
975589262557676759121380989329430387759690498518022248907122447704334289472989158845143185765417139682297342401805829336235484241275040
569756172506938792781288751162975388522137316 91
        n =  895410500789131591247526210608434064230435073148622008002020896157420207348676330194141710082894382973272467815258114647634
573148188813444435033519088264305269981539193091179788226456525968195638869981575338004436464356112945854187431758183545775809092988954
828066099460837023613853784099807700 49820739613

Original message string:
         Danylo Rudchenko

Integer encoded message:
         908933971938749480106597736930941612 63

Encrypted message( C(M) = M^e % n ):
         263405601346263980992201936373612849116188385429850988294196168254499814268075274872641065242973383562987088804062915042552921 48
484370870548308918897596179172741041330263205484351401784156551076608948734520265190458521569235837592969174809441243671359217653717409 19
228959311506453678581302710952059171 6387697

Decrypted message( M(C) = C^d % n ):
         908933971938749480106597736930941612 63

The decrypted message and the original encoded message match.
```

Рис. 1 – Результат роботи програми

Висновок: в результаті виконання лабораторної роботи було досліджено і реалізувано механізм асиметричного алгоритму шифрування RSA.