Міністерство освіти і науки України Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського» Фізико-технічний

інститут

ЛАБОРАТОРНА РОБОТА №5

з **дисципліни**

«Криптографія»

на тему: «Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем»

Виконали:

студенти 3 курсу ФТІ
Семичастний В.С
Романюк Д.О.

Перевірили:
Чорний О.
Савчук М. М.
Завадська Л. О.

Мета роботи :

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Порядок виконання роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту.

В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.

2. За допомогою цієї функції згенерувати дві пари простих чисел p,q і p_1, q_1довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $p\ q \le p_1 q_1$; p і q – прості числа для побудови ключів абонента А, p_1і q_1– абонента В.

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ *(d, p, q)* та відкритий ключ *(n, e)* . За допомогою цієї функції побудувати схеми RSA для абонентів А і В – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n), (e_1, n_1) та секретні d і d_1.

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів *A* і *B*. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання.

За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів *A* і *B*,перевірити правильність розшифрування. Скласти для *A* і *B* повідомлення з цифровим підписом і перевірити його.

5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа $0 < k < n$.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою http://asymcryptwebservice.appspot.com/?section=rsa.
Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Результати:

Згенеровані прості числа

p:895280302874095842615074978154556241406953506570865324464095581562556676789764791033271461513852837097405440078785641576225412181202975608893536571637292 3

q:821906303051684581071927747030721672734128516799450010707505696087630244194735684989525543188504005583608423321289923634530717479309287915706257145968824 1

p1:1127289766654819000218980477400399690400087302778838747980052975116525265529324690900333347276101577393014603786969234186118973452234035941866820748475168 9

q1:6987785427029564369323903722726005571511231494355653800336179098267914917651530312291312735935981943363241485666545799981820656037732324619010269073495187

Отримані відкриті та закриті ключі
e: 65537

n:735836523930240575867422361888935024601445429713437136277927311737919554545213225710366756540782323027321464473746371897183931644320245833160377450292136881165318745730660147083459518217538284751732897644368301160967780405621807965383666243280712361141369342616739416654204827684853481358429100162068738984 43

d:972327738107616062226204686506580605314328916691085280096258681302223681639615260791884906486896702231808578717769135088516845146987706015711562738533943254010415430413609100324763022004853426274161871142583461166558514007478414392299054568351449838481849451310749482397110226733378876809097716401751046387 3

e1: 65537

n1:787725900347010236004934190533712683489694468288991572568308053929516066483463081363101301214092543118370026297702302426762941446494489861053271786286975155296442355304697096614835189775935405749659618443350871391406107227031303102671345610446724090279051439004281842577247189081109189014236061409660316208431

d1:320417429107993940391222296584337286059298947703555813380624015466881903357128932099082266319259029471130936128357232984308810184791035761721746788513941419685973743408377784204859322684328335512116859107135252921376610533963850687280483979378098728245227538485590140974722722382994236180560510257266030059851

Зашифроване повідомлення та підписи:

BT: 1666

ШТ:251246739804063439030815502133296150961888908425485537386782416994507769414390591767259005500871638615989810017781094870727711714563649874483147669405056966051153815401031186682593650576872832313135186600284590632915756672003518920038172279958566720067110572992493477225137893386284907070968132710442709258751666

Підпис для А:

418364730031364240759732047745235430769790813946776453848658512716443986830081743109155158505276736934381883806191537527529832420566902144553080188867512580155383793436052355997258215254680079543901249475360222290691783191473963660106052136940333227861222505027780591549047596893197641130427934453534438435161666

3364374006915115559882375079495651085962683116066239514808929078290142862301423998525434327868183616776077266946028493601719137167337328744341550664923352385789052431216303388749466951572979225382751443887076027272093070988574522389323307359617436950252450988115394430984795540012252481929354572768771723055416666

Підпис для В:

1666

542843884111415971029636854987902091098178752738478910294711377333324579864235817364506978483019449329313808708924878844176016526902140627105064901596405416340230360299045800975546349169293314771463437418048738714473897795003772360135766195347377080557307648025159969162524754920913764897736368384643300036241666

Код программи:

```
import random
from random import randrange
import math
import Crypto.Util.number

q = 0
bit = 2048
text = 1666
print('message is 1666', '\n')


def generacia(bit, q):
        love = []
        for i in range(4):
                q = Crypto.Util.number.getPrime(bit)
                love.append(q)
        for i in love:
                proverka = proverka_deda(i, bit)
                if proverka is False:
                        generacia(bit, q)
        return love
        '''
        while len(love) < 4:
```

```
                q = (random.getrandbits(bit) |
1) + (1 << bit)
                proverka = proverka_deda(q,
50)
                if proverka is True:
                        love.append(q)
                else:
                        q =
(random.getrandbits(bit) | 1) + (1 << bit)
        return love
        '''

def generacia2_0(bit, q, q1, p, p1):
        n = q * p
        n1 = q1 * p1
        fi = (q - 1) * (p - 1)
        fi1 = (q1 - 1) * (p1 - 1)
        e = 65537
        e1 = 65537
        gc = gcd(e, fi)
        gc1 = gcd(e1, fi1)
        MMI = lambda A, n, s=1, t=0, N=0: (n
< 2 and t % N or MMI(n, A % n, t, s - A // n *
t, N or n), -1)[n < 1]

        while gc != 1:
                e = random.randint(1, fi)
                gc = gcd(e, fi)
        d = MMI(e, fi)

        while gc1 != 1:
                e1 = random.randint(1, fi1)
                gc1 = gcd(e1, fi1)
        d1 = MMI(e1, fi1)

        if n <= n1:
                return ((e, n), (d, p, q), (d, n),
(e1, n1), (d1, p1, q1), (d1, n1))
        else:
                return ((e1, n1), (d1, p1, q1),
(d1, n1), (e, n), (d, p, q), (d, n))


def gcd(a, b):
        while b != 0:
                a, b = b, a % b
        return a
def proverka_deda(chislo, bit):

        if chislo % 2 == 0:
                print('parne')
                return False

        s = 0
        d = chislo - 1
        while d and 1 == 0:
                d //= 2
                s += 1

        for i in range(bit):
                a = random.randint(2, chislo -
1)
                x = pow(a, d, chislo)
                if gcd(x, chislo) == 1:
                        if pow(x, d, chislo) == 1
or -1:
                                return True
                        for r in range(1, s - 1):
                                if pow(x, (d * (2
** r)), n) == -1:
                                        return
True
        return False


def encrypt(text, para):
        e, n = para
        C = pow(text, e, n)
        return C


def decrypt(ciphertext, para2):
        d, n = para2
        M = pow(ciphertext, d, n)
        return M


def digital(text, para2):
        d, n = para2
        S = pow(text, d, n)
        return text, S
```

```
def test_for_digital(text, para, S):
    text, s = S
    e, n = para
    if text == pow(s, e, n):
        return print('digital sign is True', '\n\n')
    else:
        return print('digital sign is False', '\n\n')

def abonent_A(text, para3, para, para2):
    e, n = para
    d, n = para2
    e1, n1 = para3
    K = encrypt(text, para3)
    S = digital(text, para2)
    return encrypt(S[1], para3), K

def abonent_B(text, s, para4, para):
    d1, n1 = para4
    e, n = para
    K = decrypt(text, para4)
    rozsh = decrypt(s, para4)
    test = test_for_digital(text, para, (K, rozsh))
    return K, rozsh
```

```
kek = generacia(bit, q)

keys = generacia2_0(bit, kek[0], kek[1], kek[2], kek[3])
print('(e, n) = ', keys[0], '\n\n', '(d,p,q) = ', keys[1], '\n\n', '(d, n) = ', keys[2], '\n\n', '(e1, n1) = ', keys[3], '\n\n', '(d1,q1,p1) = ', keys[4], '\n',)
zash = encrypt(text, keys[0])
rozsh = decrypt(zash, keys[2])
print('Ciphertext = ', zash, '\n\n','Plaintext = ', rozsh, '\n')
sign = digital(text, keys[2])
#test_for_digital(text, keys[0], sign)
send = abonent_A(text, keys[3], keys[0], keys[2])
get = abonent_B(send[1], send[0] ,keys[5], keys[0])
print('abonent_A = ', send, '\n\n')
print('secret message from abonent_A =', send[1], '\n\n')
print('decrypted message from abonent_A =', get[0], '\n\n')
print('abonent_B = ', get, '\n\n')
```

Висновки:

Під час данного комп’ютерного практикуму, ми ознайомились з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Також, використовуючи криптосистему типу RSA, організували канал засекреченого зв’язку й електронний підпис, ознайомились із протоколом розсилання ключів.