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

групи ФБ-72

Король Михайло, Степанець Антон

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

Мета та основні завдання 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 і рекомендації щодо 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 p1, q1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤ p1q1 ; p і q – прості числа для побудови ключів абонента A, p1 iq1 – абонента B. 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 A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (e1, n1) та секретні d і d1 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 и 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.

Хід роботи:

Генерація випадкових p і q:

p	q
3016637147122873629209122446984212825 49	17144001812903887759074637494760512990 5
3016637147122873629209122446984212825 51	17144001812903887759074637494760512990 7
3016637147122873629209122446984212825 53	17144001812903887759074637494760512990 9
3016637147122873629209122446984212825 55	17144001812903887759074637494760512991 1
3016637147122873629209122446984212825 57	17144001812903887759074637494760512991 3
3016637147122873629209122446984212825 59	17144001812903887759074637494760512991 5
3016637147122873629209122446984212825 61	17144001812903887759074637494760512991 7

30166371471228736292091224469842128256 3	171440018129038877590746374947605129919
30166371471228736292091224469842128256 5	171440018129038877590746374947605129921
30166371471228736292091224469842128256 7	171440018129038877590746374947605129923
30166371471228736292091224469842128256 9	171440018129038877590746374947605129925
171440018129038877590746374947605129905	171440018129038877590746374947605129927
	171440018129038877590746374947605129929
	171440018129038877590746374947605129931
	…
	171440018129038877590746374947605129995
	171440018129038877590746374947605129997
	171440018129038877590746374947605129999
	171440018129038877590746374947605130001

Обрані

P = 171440018129038877590746374947605129905

Q = 171440018129038877590746374947605130001

Відкритий текст:

961619429875513990770820127159101470592140612470906184790274618392469935113 67

Зашифрований закритий ключ:

521ba069ac2f10f4632bdd22afad537572b22ce2c11ef655a554e04fe4463429

Цифровий підпис:

ac49d6323dd35a88ef937385bde757490542f6d2477d9e7f073da34f5a58e096

Код програми:

```
from random import randint as rd
from math2 import Power_by_modulus
primary  = [2, 3, 5, 7,11,13, 17, 19, 23, 29, 31]
def ExEuclid(q, w):
        if q == 0:
                return (w, 0, 1)
        else:
                gcd, first, second = ExEuclid(w % q, q)
                return (gcd, second - (w // q) * first, first)
```

```
def Trivial_check(dec_number):
        decimal_number = dec_number
        global primary
        for i in primary:
                if decimal_number % i == 0:
                        return False
        return True



class User():
        def __init__(self):
                self.__p = self.GeneratePN()
                self.__q = self.GeneratePN()
                print("aaaaaa",self.__p,self.__q)
                self.n = self.__p * self.__q
                self.__fi = self.FiFunction()
                self.GenerateKeyPair()

        def GenerateRand(self,length = 128):
                binary_number = ""
                for  i in range(length - 2):
                        binary_number += str(rd(0,1))
                binary_number ="1" + binary_number[:] + "1"
                return binary_number

        def FiFunction(self):
                return (self.__q - 1)*(self.__p - 1)

        def GenerateKeyPair(self):
                self.e = rd(2,self.__fi - 1)

                while ExEuclid(self.e,self.__fi)[0] != 1:
                        self.e = rd(2,self.__fi - 1)
                self.__d = ExEuclid(self.e,self.__fi)[1]%self.__fi
                print(str(hex(self.e)[2:]),"  ASDASDA   ",str(hex(self.n)[2:]))
```

```
    def Encryption(self,PlainText,receiver_e,receiver_n):
        return Power_by_modulus(PlainText,receiver_e,receiver_n)

    def DigitalSignature(self,PlainText):
        return Power_by_modulus(PlainText,self.__d,self.n)

    def SignatureVerification(self,PlainText,Digest,sender_e,sender_n):
        return PlainText == Power_by_modulus(Digest,sender_e,sender_n)

    def Decryption(self,CipherText):
        return Power_by_modulus(CipherText,self.__d,self.n)

    def SendKey(self,PainText,receiver_e,receiver_n,sender_e,sender_n):
        return
(self.Encryption(PainText,receiver_e,receiver_n),self.Encryption(self.DigitalSignature(PainText),receiver_e,receive
r_n),(sender_e,sender_n))

    def ReceiveKey(self,CipherText,DigitalSignature,senders_public):
        PainText = self.Decryption(CipherText)
        print(PainText)
        DS =
self.SignatureVerification(PainText,self.Decryption(DigitalSignature),senders_public[0],senders_public[1])
        if DS:
            return (PainText,DS)
        return None

    def PrimaryTestMillera_Rabina(self,dec_number):
        k = 0
        decimal_number = dec_number - 1
        decimal = decimal_number + 1
        d,s = 0,0
        while not(decimal_number % 2  == 1):
            decimal_number  = decimal_number // 2
            s = s + 1
        d = int(decimal_number)
        while k != 5:
```

```
                x = rd(2,decimal - 1)
                if ExEuclid(decimal,x)[0] != 1:
                        return False
                if Power_by_modulus(x, d, decimal) in [-1 % decimal,1]:
                        k += 1
                        continue
                else:
                        check = False
                        for i in range(1,s):
                                r = Power_by_modulus(x,d * pow(2,i), decimal)
                                if r  == 1:
                                        return False
                                if r  == -1 % decimal:
                                        check = True
                                        break
                        if not check :
                                return False
                k += 1
        return decimal


    def GeneratePN(self):
        p = int("".join(self.GenerateRand()),2) - 2
        while True:
                p += 2
                print(p)
                #if not Trivial_check(p):
                #       continue
                advanced_p = self.PrimaryTestMillera_Rabina(p)
                if  advanced_p :
                        return advanced_p


a = User()
Site_public_n =
int("D499BB1FEA2F8BB3DD8FF84D056BC297DF552EECCC01CD607793CEDD534DD3C7",16)
```

```
print(Site_public_n,"       DD")

Site_public_e = int('10001', 16)

while a.n > Site_public_n:


        a = User()


tpl =
a.SendKey(772236207544142343524988776290862243162112092754627934242299,Site_public_e,Site_public_n,
a.e,a.n)

print("AAAAAAAAAAAAAAA   ",str(hex(tpl[0])[2:]))

print("DS    ",str(hex(tpl[1])[2:]))

#print(tpl)

#print(b.ReceiveKey(tpl[0],tpl[1],tpl[2]))

#print(b.Decryption(a.Encryption(2,b.e,b.n)))
```

Висновок: у ході комп’ютерного практикуму було набуто навичок роботи з числами великої розрядності, написання тестів перевірки чисел на простоту та методів генерації ключів для асиметричної криптосистеми RSA. Набуто навичок побудови цифрового підпису на основі криптосистеми RSA. На основі RSA було побудовано модель протоколу для розсилання ключів по відкритих каналах.