МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

КРИПТОГРАФІЯ

Комп’ютерний практикум №5

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

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

Виконали:
Студентки групи ФБ-71
Гресь В.В.
Нацвін К.А.

Київ 2019

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_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p_1q_1$; *p* і *q* – прості числа для побудови ключів абонента *A*, p_1, q_1 – 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), (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* 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. Перевірити роботу програм для випадково обраного ключа $0<k<n$

Опис роботи та основні труднощі

Найважчою частиною у виконанні даної лабораторної було згенерувати число довжиною 256 біт, яке мало бути простим. Для цього використаний тест перевірки на простоту Міллера-Рабіна. Далі, за допомогою цієї функції було згенеровано два числа – p та q і p_1q_1 , які використовувались для написання функцій для зашифрування, розшифрування, створення ЦП, його підписання та перевірки, а також для надсилання та отримання ключів. На етапах розробки цих функцій, особливих труднощів не виникло.

Кандидати p, q, p_1, q_1, що не пройшли перевірку (перші 20):

259376944164536246619858461562494577018
298626401231583149371999914639665279362
139363098906306842984032098623713966650
259970373090755644626439781151036070912
179760125119683361616057361838643729853
163251713802645054258289838186598239299
409915121780743897280549046458851480 34
140647304156977176024055909914902712828
109852671881424993604426133561897890778
958746125632652175436074242503358071 93
110898353795221335146365948077330536890
565911665905510977068199167642136541 45
166583004309505438760747752949968037222
278265498134264878605530522174407879264
112694636612660585706079159265291113717
270952726040516391654086566464221472508
251180544743888740179647344359592781821
326535042941222580631235016178076638744
127507427079295630047248211486710084017
148223949820417627364878557596429019881

Кандидати p, q, p_1, q_1, що пройшли перевірку:

217428126797360283093821490491456127761,
228018264418052102410310024851487424479,
233164296390046700360479204833711926929,
247396739840989874524999961915733353047

Використовуване чисельне значення e:
65537
Використовуване чисельне значення n:
495775841080022570643736598960787269858021143579361260549139424456362628615 19

Використовуване чисельне значення BT:
64567844567
Використовуване чисельне значення ШТ:
713186706168022881977497812680704173533255734686945659148667764877048358568 9
Використовуване чисельне значення ЦП:
577748224208402568504881375591769679709418461480848361920449818290094018542 9

Код програми

```
import random
import sys
sys.setrecursionlimit(1500)
dec = 64567844567
print(hex(dec)[2:].upper())


class RSA_UI:
    def __init__(self):
        self.e = 65537
        self.e1 = 3
        print('Hexed e: ', hex(self.e))
        self.keys = self.number_generator()
        self.keys.sort()
        print(self.keys)
        self.needed = self.check_nums(self.keys[0], self.keys[1], self.keys[2], self.keys[3])
        self.d = self.inverse(self.e, self.needed[3])


    @staticmethod
    def miller_rabin(n, k):
```

```
        q, r = 0, n - 1
        while r % 2 == 0:
            q += 1
            r //= 2
        for _ in range(k):
            a = random.randrange(2, n - 1)
            x = pow(a, r, n)
            if x == 1 or x == n - 1:
                continue
            for _ in range(q - 1):
                x = pow(x, 2, n)
                if x == n - 1:
                    break
            else:
                return False
        return True

    def number_generator(self):
        numbers = []
        while len(numbers) < 4:
            new_num = random.getrandbits(128)
            print(new_num)
            if self.miller_rabin(new_num, 50) is True:
                numbers.append(new_num)
        return numbers

    @staticmethod
    def check_nums(n1, n2, n3, n4):
        if n1 * n2 <= n3 * n4:
            n = n1 * n2
            m = n3 * n4
            phi = (n1 - 1) * (n2 - 1)
            phi_1 = (n3 - 1) * (n4 - 1)
            print('Hexed m: ', hex(m)[2:].upper())
            print('Hexed p*q: ', hex(n)[2:].upper())
            print('p*q: ', n)
            return n, m, phi_1, phi

    def encrypt(self, decimal):
        n = self.needed[0]
        e = self.e
        return pow(decimal, e, n)

    @staticmethod
    def inverse(a, m):
        m_0 = m
        z = 0
        fin = 1

        if m == 1:
            return 0

        while a > 1:
            tmp = a // m
            l = m
            m = a % m
            a = l
            l = z
            z = fin - tmp * z
            fin = l

        if fin < 0:
            fin = fin + m_0

        return fin

    def decrypt(self):
        enc = self.encrypt(dec)
        return pow(enc, self.d, self.needed[0])

    def sign(self, decimal):
        s = pow(decimal, self.d, self.needed[0])
        m = pow(s, self.e, self.needed[0])
        return s, m

    def verify_sign(self, decimal):
        return pow(self.sign(decimal)[0], self.e, self.keys[0])

    def send_key(self):
        k = 3
        k1 = pow(k, self.e, self.needed[1])
        s = pow(k, self.d, self.needed[0])
        s1 = pow(s, self.e, self.needed[1])
        return k1, s1, s, self.needed[0]

    def receive_key(self):
        received = self.send_key()
        k = pow(received[0], self.inverse(self.e, self.needed[3]), self.needed[0])
        s = pow(received[1], self.inverse(self.e, self.needed[3]), self.needed[0])
        return k, s
```

```
RSA start = RSA UI()
encrypted = RSA start.encrypt(dec)
print ('Encrypted message:', encrypted)
print('Sign of message', RSA_start.sign(dec)[0])
print('Encrypted message:',hex(encrypted)[2:].upper())
print('Message:', dec, '\nDecryption result:', RSA_start.decrypt())
sent = RSA_start.send_key()
rec = RSA start.receive key()
print('Sign of message', hex(RSA_start.sign(dec)[0])[2:].upper())
print(hex(RSA start.e)[2:])
print(hex(RSA_start.verify_sign(dec))[2:].upper())
print('Sent: ', hex(sent[0])[2:].upper(), hex(sent[1])[2:].upper(), hex(sent[2])[2:].upper(),
hex(sent[3])[2:].upper())
print('Received Key: ', hex(rec[0])[2:].upper(), hex(rec[1])[2:].upper())
```

Висновок

Під час даного практикуму ми ознайомилися з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA, організували канал засекреченого зв’язку й електронний підпис, ознайомилися з протоколом розсилання ключів