Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Фізико-технічний інститут

Лабораторна робота №5

З предмету «Криптографія»

Виконали:
Студенти 3 курсу,
ФТІ, групи ФБ-72
Курт Олег, Вовчук Роман

Київ 2019

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 1 1 p , q довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $p1q1 \leq pq$; p і q – прості числа для побудови ключів абонента A, 1 p і q1 – 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) , (,) 1 n1 e 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. Перевірити роботу програм для випадково обраного ключа $n.< k <0$

Труднощі та етапи розробки програмного коду:

Перш за все було розроблено функцію перевірки числа на простоту. Для цього був використаний тест Мілера-Рабіна. Першою перешкодою була необхідність швидко піднести число в степінь по модулю, так як звичайна математична операція піднесення числа в степінь довго працює з великими числами. Тому ми створили функцію за схемою Горнера, яка вирішує це питання.

Далі ми отримали пари простих чисел p1, q1, p, q та написали функції за шифрування, розшифрування, підписання ЦП, його перевірки, та функції відправки та отримання ключів. На цих етапах проблем не було.

Результати:

Абонент A

p = 227496347407064019521732178144932707283

q = 182382548131183261160636137166502733013

n = 41491363530637241834486943007250722141911620169365798894849569950592629633679

e = 166100449583627074025940199223872978703321171146514462539015552393867305239991

d = 74980969642395105606827116108950408375913385570011224050854526203054993657783

Відкритий текст

m = 28506493915693425373655957622674087292219892102801793724598822776937957525527

Шифрований текст

c = 20422718738383845175567856483096639345901000163398085081089196557281160771925

Підпис...

s = 64937767048446149325220838539842771250258730279081795412835242078373947043 83

Абонент Б

p = 32375115586735682249029991948507661359 9

q = 335016179310341735937440217300099527911

n = 10846187528598880922390984365307808422172699817513503639314451293372646266168 9

e = 285319159435531029853753838405458929145034897401457272691363933266808299715 53

d = 31629018286948505646613136870347809579100528582179925981524100272486267943777

Відкритий текст

m = 54978701911538728362541687270736707241314110132456772973047082215955536407057

Шифрований текст

c = 93419014414490386050645513491903249132097530875074455786252537369098781068 19

Підпис...

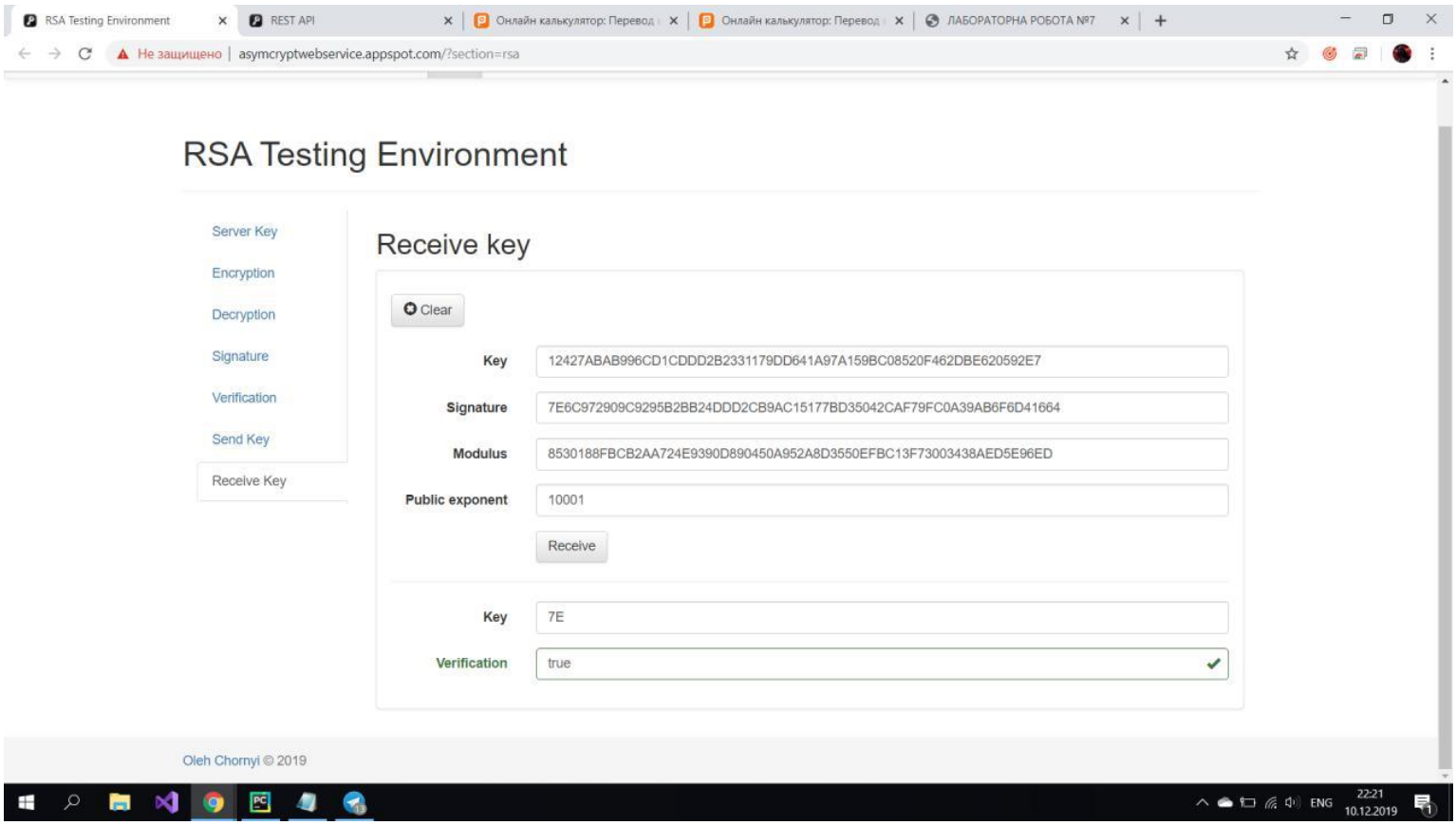
s = 13816486797409006044330482563569673566023222887072542105314737666247971301245

Відправка ключа

к1 = 15403479317446713232930234899060839475880285567089824953078197436006291304009

s1 = 51847077957643887416334547237039794446263914992774670656952368961647944133505

Результати з сайта:



Програмний код:

import random	xx = per - xx * q	if reverse(x, p, 0) > 1:
import math	if j == 0:	return -p
def quick(b, a, m):	return a	else:

```
    w = "{0:b}".format(a)
    i = 0
    y = 1
    while i != len(w):
        y = (y ** 2) % m
        y = (y * (b ** int(w[i]))) % m
        i += 1
    return y
def reverse(a, b, j):
    c = b
    x = 1
    xx = 0
    while b != 0:
        q = a // b
        per = a
        a = b
        b = per % b
        per = x
        x = xx
    else:
        if x < 0:
            x = c + x
        if a == 1:
            return x
        else:
            print("revers element don't exist")
        return -a
def prime(p):
    k1 = math.log(p, 2)
    k = int(k1)
    if k1 % k != 0.0:
        k += 1
    d = p - 1
    s = 0
    while d % 2 == 0:
        s += 1
        d //= 2
    for i in range(k):
        x = random.randint(2, p - 1)
        if quick(int(x), int(d), p) == 1:
            return p
        if quick(int(x), int(d), p) == -1:
            return p
        else:
            r = 0
            while r != s:
                xr = quick(x, int(d) * (2 ** r), p)
                if xr == -1:
                    return p
                elif xr == 1:
                    return -p
                else:
                    r += 1
            if i < k:
                i += 1
            else:
                return -p
    return -p
def generate_key_pair(bit):
    big = '1'
```

```
    i = 0
    while i != bit-2:
        big += str(random.randint(0, 1))
        i += 1
    big += '1'
    p = -1 * int(big, 2)
    while p < 0:
        p = -1 * p + 2
        p = prime(p)
        if p > 0:
            break
print('p = ' + str(p))
    big = '1'
    i = 0
    while i != bit-2:
        big += str(random.randint(0, 1))
        i += 1
    big += '1'
    q = -1 * int(big, 2)
    while q < 0:
        q = -1 * q + 2
        q = prime(q)
        if q > 0:
            return d, n, e, m, c
    m = random.randint(0, n-1)
    print('m = ' + str(m))
    c = quick(m, e, n)
    print('c = ' + str(c))
    return 0, n, e, m, c
def decrypt(d, c, n):
    print('Decrypt...')
    m = quick(c, d, n)
    print('m = ' + str(m))
    return 1
def sign(m, d, n):
    print('Sign...')
    s = quick(m, d, n)
    print('s = ' + str(s))
    return s
def verify(s, e, n):
    print('Verify...')
    m = quick(s, e, n)
    print('m = ' + str(m))
    return m
def receive_key(k1, d1, n1, s1, e, n):
    print('Receive_Key...')
    print('Upppps.....')
    print('\n'*25)
    arr1 = encrypt(0, 0)
    arr2 = encrypt(0, 0)
s1 = sign(arr1[3], arr1[0], arr1[1])
verify(s1, arr1[2], arr1[1])
decrypt(arr1[0], arr1[4], arr1[1])
print('\n\n')
s2 = sign(arr2[3], arr2[0], arr2[1])
verify(s2, arr2[2], arr2[1])
decrypt(arr2[0], arr2[4], arr2[1])
print('\n\n')
sk = send_key(arr2[2], arr1[1], arr2[1], arr1[0])
receive_key(sk[0], arr2[0], arr2[1], sk[1], arr1[2],
arr1[1])
```

```
            break
        print('q = ' + str(q))
        n = p * q
        print('n = ' + str(n))
        fi = (p - 1) * (q - 1)
        e = random.randint(2, fi - 1)
        while reverse(e, fi, 0) != 1:
            e = random.randint(2, fi - 1)
        print('e = ' + str(e))
        d = reverse(e, fi, 1)
        print('d = ' + str(d))
        return n, e, d
    def encrypt(e, n):
        print('Encrypt...')
        if e == 0 and n == 0:
            n, e, d = generate_key_pair(128)
            m = random.randint(0, n - 1)
            print('m = ' + str(m))
            c = quick(m, e, n)
            print('c = ' + str(c))
```

```
        k = quick(k1, d1, n1)
        print('k = ' + str(k))
        s = quick(s1, d1, n1)
        print('s = ' + str(s))
        print('Check...')
        k = quick(s, e, n)
        print('k = ' + str(k))
        return k
    def send_key(e1, n, n1, d):
        print('Send_Key...')
        k = random.randint(1, n-1)
        print('k = ' + str(k))
        s = quick(k, d, n)
        s1 = quick(s, e1, n1)
        print('s1 = ' + str(s1))
        k1 = quick(k, e1, n1)
        return k1, s1
    arr1 = encrypt(0, 0)
    arr2 = encrypt(0, 0)
    while arr2[1] < arr1[1]:
        print('\n\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.