Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

По дисциплине «КМЗИ»

Тема: “Программная реализация ЭЦП”

Выполнил:

Студент 3 курса

Группы ИИ-21

Романко Н.А.

Проверила:

Хацкевич А.С.

Брест 2023

Цель:Cоздать программу, которая реализует учебный вариант

схем ЭЦП, используя алго

1. Метод: Алгоритм Рабина

Код клиента:

from socket import \*

import random

import os

client = socket(

    AF\_INET, SOCK\_STREAM

)

client.connect(

    ('localhost', 7000)

)

def send():

    message = input("Введите текст сообещния: ")

    client.send(message.encode('utf-8'))

def recive():

    print("Ожидание сообщения")

    msg = client.recv( 8192 ).decode('utf-8')

    print(f'SERVER MESSAGE:\n\t{msg}')

def message\_to\_bits(message):

    bits = ''.join(format(ord(char), '08b') for char in message)

    return bits

def is\_prime(num):

    if num <= 1:

        return False

    if num <= 3:

        return True

    if num % 2 == 0 or num % 3 == 0:

        return False

    i = 5

    while i \* i <= num:

        if num % i == 0 or num % (i + 2) == 0:

            return False

        i += 6

    return True

def generate\_prime(min\_value, max\_value):

    prime = random.randint(min\_value, max\_value)

    while not (is\_prime(prime) and prime % 4 == 3):

        prime = random.randint(min\_value, max\_value)

    return prime

def create\_n():

    p = generate\_prime(30, 10000)

    print("p:", p)

    client.send(str(p).encode('utf-8'))

    q = generate\_prime(30, 10000)

    print("q:", q)

    client.send(str(q).encode('utf-8'))

    n = p \* q

    print("n:", n)

    client.send(str(n).encode('utf-8'))

    return p, q, n

def main():

    p, q, n = create\_n()

    message = input("Введите сообщение для шифровки: ")

    client.send(message.encode('utf-8'))

    m = [ord(char) for char in message]

    m\_binary = ''.join(format(char, '08b') for char in m)

    extended\_m\_binary = m\_binary + m\_binary

    m = int(extended\_m\_binary, 2)

    c = (m^2) % n

    client.send(str(c).encode('utf-8'))

while True:

    task = input("Выберете действие:\n\t 1-отправить сообщение\n\t 2-получить сообщение\n\t 3-лаба\n")

    match task:

        case "1":

            send()

        case "2":

            recive()

        case "3":

            main()

        case "9":

            pass

        case "e":

            client.close()

            os.system("cls")

            break

        case \_:

            print("Нет такого лействия")

Код сервера:

import os

from socket import \*

server = socket(

    AF\_INET, SOCK\_STREAM  #параметры подключения

)

server.bind(

    ('localhost', 7000)  #адресс и сокет сервера

)

server.listen(2)  #количество подключений

print("SERVER IS WORKING\n")

user, addr = server.accept()  #после подключения, выполнять далее

print(f"CONNECTED:\n\t{user}\n\t{addr}\n")

user.send('Connected successful!'.encode('utf-8'))  #отправить сообщение

def send():

    message = input("Введите текст сообещния: ")

    user.send(message.encode('utf-8'))

def recive():

    print("Ожидание сообщения")

    msg = user.recv( 8192 ).decode('utf-8')

    print(f'SERVER MESSAGE:\n\t{msg}')

def extended\_gcd(a, b):

    if a == 0:

        return (b, 0, 1)

    else:

        gcd, x, y = extended\_gcd(b % a, a)

        return (gcd, y - (b // a) \* x, x)

def find\_a\_b\_for\_equation(p, q):

    gcd, a, b = extended\_gcd(p, q)

    return a, b

def message\_to\_bits(num, bit\_length=8):

    if num >= 0:

        bits = bin(num)[2:].zfill(bit\_length)

    else:

        # Преобразование отрицательного числа в дополнительный код

        num = abs(num)

        bits = bin(num)[2:]

        if len(bits) < bit\_length:

            bits = '0' \* (bit\_length - len(bits)) + bits

        inverted = ''.join('1' if bit == '0' else '0' for bit in bits)

        inverted = inverted[-bit\_length:]

        carry = 1

        result = ''

        for bit in inverted[::-1]:

            if bit == '0' and carry == 1:

                result = '1' + result

                carry = 0

            elif bit == '1' and carry == 1:

                result = '0' + result

            else:

                result = bit + result

        bits = result

    return bits

def main():

    p = int(user.recv( 8192 ).decode('utf-8'))

    print("p: ", p)

    q = int(user.recv( 8192 ).decode('utf-8'))

    print("q: ", q)

    n = int(user.recv( 8192 ).decode('utf-8'))

    print("n: ", n)

    m = (user.recv( 8192 ).decode('utf-8'))

    c = int(user.recv( 8192 ).decode('utf-8'))

    print("c: ", c)

    a, b = find\_a\_b\_for\_equation(p, q)

    r = (c \* (p + 1) // 4) % p

    s = (c \* (q + 1) // 4) % q

    x = (a \* p \* r + b \* q \* s) % p

    y = (a \* p \* r - b \* q \* s) % q

    m1 = x; m2 = -x; m3 = y; m4 = -y

    m1\_bits = message\_to\_bits(m1)

    m2\_bits = message\_to\_bits(m2)

    m3\_bits = message\_to\_bits(m3)

    m4\_bits = message\_to\_bits(m4)

    half\_length\_m1 = len(m1\_bits) // 2

    half\_length\_m2 = len(m2\_bits) // 2

    half\_length\_m3 = len(m3\_bits) // 2

    half\_length\_m4 = len(m4\_bits) // 2

    left\_half\_m1 = m1\_bits[:half\_length\_m1]

    right\_half\_m1 = m1\_bits[half\_length\_m1:]

    left\_half\_m2 = m2\_bits[:half\_length\_m2]

    right\_half\_m2 = m2\_bits[half\_length\_m2:]

    left\_half\_m3 = m3\_bits[:half\_length\_m3]

    right\_half\_m3 = m3\_bits[half\_length\_m3:]

    left\_half\_m4 = m4\_bits[:half\_length\_m4]

    right\_half\_m4 = m4\_bits[half\_length\_m4:]

    matching\_half = None

    if left\_half\_m1 == right\_half\_m1:

        matching\_half = left\_half\_m1

    elif left\_half\_m2 == right\_half\_m2:

        matching\_half = left\_half\_m2

    elif left\_half\_m3 == right\_half\_m3:

        matching\_half = left\_half\_m3

    elif left\_half\_m4 == right\_half\_m4:

        matching\_half = left\_half\_m4

    if matching\_half is not None:

        # Преобразование половины в десятичное число

        matching\_half\_decimal = int(matching\_half, 2)

        # Получение ASCII символа для десятичного числа

        message = chr(matching\_half\_decimal)

        print(f"Decoded Message: {message}")

    else:

        print(f"Decoded Message: {m}")

while True:

    task = input("Выберете действие: \n\t 1-отправить сообщение\n\t 2-получить сообщение\n\t 3-лаба\n")

    match task:

        case "1":

            send()

        case "2":

            recive()

        case "3":

            main()

        case "9":

            pass

        case "e":

            server.close()

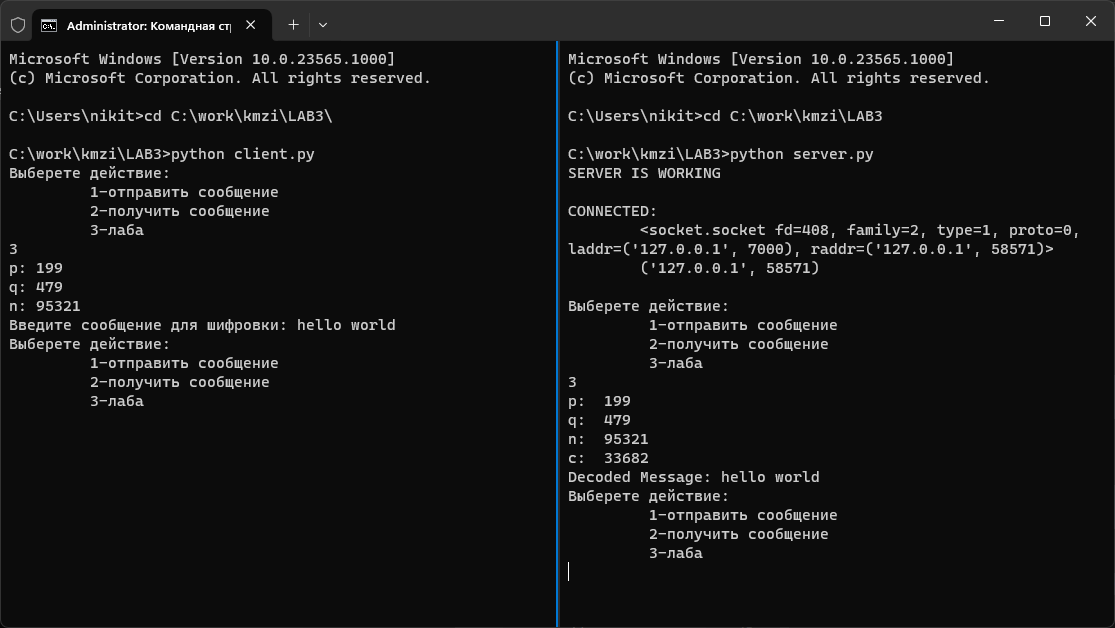
            os.system("cls")

            break

        case \_:

            print("Нет такого действия")

Вывод программы:



Вывод: изучил основные алгоритмы электронной цифровой подписи.