Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Криптографические методы защиты информации»

Тема: «Алгоритмы обмена ключами»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Литвинюк Т. В.

**Проверил:**

Хацкевич А. С.

Брест 2023

**Цель:** изучить алгоритмы обмена ключами. Практически реализовать алгоритмы обмена ключами.

**Ход работы:**

**Вариант**: EKE на основе RSA, RC-4

Код сервера:

import socket

from threading import Thread

clients = []

def resend\_messages(client: socket.socket):

    while True:

        try:

            data = client.recv(512)

        except ConnectionResetError:

            server.close()

            return

        else:

            if "!exit" in data.decode():

                server.close()

                return

            for cl in clients:

                if cl != client:

                    cl.send(data)

if \_\_name\_\_ == '\_\_main\_\_':

    server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server.bind(('localhost', 8080))

    server.listen(2)

    while len(clients) != 2:

        client = server.accept()[0]

        print("user connected")

        clients.append(client)

        Thread(target=resend\_messages, args=(client,)).start()

    clients[0].send("!ready".encode())

    clients[1].send("!ready".encode())

Код клиента:

import socket

from threading import Thread

from random import randint

from encrypting import generate\_rsa\_keys, rc4

def send\_message(key):

while True:

message = input()

if '!exit' in message:

client.send(message.encode())

return

client.send(rc4(message, key).encode())

def receive\_message(key):

while True:

message = client.recv(512).decode()

print(rc4(message, key))

if \_\_name\_\_ == '\_\_main\_\_':

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect(('localhost', 8080))

message = client.recv(512).decode()

if "!ready" in message:

eA, dA, NA = generate\_rsa\_keys()

client.send(f"{eA} {NA}".encode())

eB, NB = map(int, client.recv(512).decode().split()) # получение открытого ключа от другого пользователя

keyA = randint(100000, 1000000)

client.send(f"{pow(keyA, eB, NB)}".encode())

t = client.recv(512).decode()

keyB = pow(int(t), dA, NA)

print(f"Обмен ключами произошел. {keyA=}, {keyB=}")

send\_message\_thr = Thread(target=send\_message, args=(keyB,))

recv\_message\_thr = Thread(target=receive\_message, args=(keyA,))

send\_message\_thr.start()

recv\_message\_thr.start()

send\_message\_thr.join()

client.close()

Код функций для шифрования:

import random

# Функция для проверки простоты числа

def is\_prime(n):

    if n <= 1:

        return False

    if n <= 3:

        return True

    if n % 2 == 0 or n % 3 == 0:

        return False

    i = 5

    while i \* i <= n:

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

            return False

        i += 6

    return True

# Функция для поиска наибольшего общего делителя

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a

def rc4(plaintext, key):

    # Инициализация S-блока

    S = list(range(256))

    j = 0

    for i in range(256):

        j = (j + S[i] + (key & 0xFF)) % 256

        S[i], S[j] = S[j], S[i]

    # Инициализация переменных

    i = 0

    j = 0

    encrypted = []

    for char in plaintext:

        i = (i + 1) % 256

        j = (j + S[i]) % 256

        S[i], S[j] = S[j], S[i]

        k = S[(S[i] + S[j]) % 256]

        encrypted\_char = ord(char) ^ k

        encrypted.append(encrypted\_char)

    # Преобразование зашифрованных байтов в строку

    encrypted\_text = ''.join([chr(byte) for byte in encrypted])

    return encrypted\_text

# Функция для нахождения обратного по модулю (расширенный алгоритм Евклида)

def mod\_inverse(e, phi):

    d = 0

    x1, x2 = 0, 1

    y1, y2 = 1, 0

    while phi != 0:

        quotient = e // phi

        e, phi = phi, e % phi

        x1, x2 = x2 - quotient \* x1, x1

        y1, y2 = y2 - quotient \* y1, y1

    if x2 < 0:

        x2 += phi

    return x2

def generate\_rsa\_keys():

    while True:

        p = random.randint(1000000, 10000000)

        if is\_prime(p):

            break

    while True:

        q = random.randint(1000000, 10000000)

        if is\_prime(q) and q != p:

            break

    # Вычисление N и функции Эйлера phi(N)

    N = p \* q

    phi\_N = (p - 1) \* (q - 1)

    # Выбор числа e (обычно простого, но не обязательно)

    e = random.randint(2, phi\_N)

    while gcd(e, phi\_N) != 1:

        e = random.randint(2, phi\_N)

    # Вычисление числа d, обратного e по модулю phi(N)

    d = mod\_inverse(e, phi\_N)

    # Вывод открытого и закрытого ключей

    return e, d, N

if \_\_name\_\_ == '\_\_main\_\_':

    # print(ord(rc4('hello', 15131)))

    # print(rc4(rc4('hello', 456672).encode(), 456672).decode())

    print(rc4(rc4('hello', 757636667), 757636667))

    print(rc4(rc4('hello', 18654791), 18654791))



**Вывод:** в ходе лабораторной работы я изучил алгоритмы обмена ключами, а также реализовал клиент-серверное приложение, позволяющее обмениваться зашифрованными сообщениями.