Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

Расчетно-графическая работа по дисциплине

«Защита информации»

на тему «Доказательство с нулевым знанием»

«Вариант 3»

Выполнил: студент 4 курса

ИВТ, гр. ИП-813

Стояк Ю.К.

Проверил: Старший преподаватель кафедры ПМиК

Дьячкова Ирина Сергеевна

.

Новосибирск 2021

**Содержание**

[1. Постановка задачи 3](#_Toc52197681)

[2. Теоретические сведения 4](#_Toc52197682)

[3. Ход работы 9](#_Toc52197683)

[4. Скриншоты работа программы 1](#_Toc52197686)1

[5. Список литературы 1](#_Toc52197686)3

[6. Листинг кода](#_Toc52197688) 14

**Постановка задачи**

Необходимо написать программу, реализующую алгоритм доказательства с нулевым знанием Фиата-Шамира

Для выполнения этого варианта задания необходимо разработать клиент-серверное приложение с авторизацией по протоколу Фиата-Шамира. Открытые ключи с соответствующими логинами должны храниться в файле (или базе данных) на сервере, клиентское приложение при этом не должно отправлять на сервер никаких закрытых данных, закрытый ключ нигде не хранится и используется исключительно для осуществления работы протокола с клиентской стороны. Все открытые параметры системы рассылаются сервером при установке соединения с клиентом.

**Теоретические сведения**

Одним из наиболее известных протоколов идентификации личности с помощью доказательства с нулевым знанием является [протокол, предложенный Амосом Фиатом и Ади Шамиром,](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%A4%D0%B8%D0%B0%D1%82%D0%B0-%D0%A8%D0%B0%D0%BC%D0%B8%D1%80%D0%B0)стойкость которого основывается на сложности извлечения квадратного корня по модулю достаточно большого составного числа n, факторизация которого неизвестна.  
  
Предварительно, перед самим доказательством доверенный центр T выбирает и публикует модуль достаточно большого числа n = p\*q, разложить на множители которое трудно. При этом p, q – простые числа и держатся в секрете. Каждый пользователь A выбирает секретное s из интервала (1, n−1) взаимно простое с n. Затем вычисляется открытый ключ v = s^2 (mod n).

Полученное v регистрируется центром доверия в качестве открытого ключа пользователя A, а значение s является секретом A. Именно знание этого секрета s необходимо доказать A стороне В без его разглашение за t раундов. Каждая аккредитация состоит из следующих этапов:  
1. А выбирает случайное r из интервала (1, n−1) и отсылает x = r^2 (mod n) стороне B.  
2. B случайно выбирает бит e (0 или 1) и отсылает его A.  
3. А вычисляет y = r\*s^e (mod n) и отправляет его обратно к B.  
4. Сторона B проверяет равенство y^2 ≡ x\*v^e (mod n). Если оно верно, то происходит переход к следующему раунду протокола, иначе доказательство не принимается.

Выбор е из множества предполагает, что если сторона А действительно знает секрет, то она всегда сможет правильно ответить, вне зависимости от выбранного e. Допустим, что А хочет обмануть B, выбирает случайное r и отсылает x = r^2 / v, тогда если е=0, то А удачно возвращает B y = r, в случае же е=1, А не сможет правильно ответить, т.к. не знает s, а извлечь квадратный корень из v по модулю n достаточно сложно.  
  
Вероятность того, что пользователь А не знает секрета s, но убеждает в обратном проверяющего B будет оцениваться вероятностью равной p =2^(–t), где t – число аккредитаций. Для достижения высокой достоверности его выбирают достаточно большим (t = 20 − 40). Таким образом, B удостоверяется в знании А тогда и только тогда, когда все t раундов прошли успешно.  
  
Для того чтобы этот протокол корректно выполнялся, сторона А никогда не должна повторно использовать значение x. Если бы А поступил таким образом, а B во время другого цикла отправил бы А на шаге 2 другой случайный бит r, то B бы имел оба ответа А. После этого B может вычислить значение s, и ему будет известен секретный ключ Алисы.

**Ход работы**

Реализованные методы в программе rgr.py

**def check\_prime(p):**

Функция проверяет число на простоту

**def generate\_prime(left, right):**

Функция возвращает случайное простой число из передаваемой границы

**def generate\_coprime(p):**

Функция возвращает случайное число взаимно-простое с передаваемым

**def pow\_module(a, x, p):**

Функция возводит в степень по модулю

**def gcd(a, b):**

Функция - Алгоритм Евклида

**def gen():**

Функция – Генерирует n из простых p и q

**def start(client)**Функция, начинает цикл аккредитации пользователя

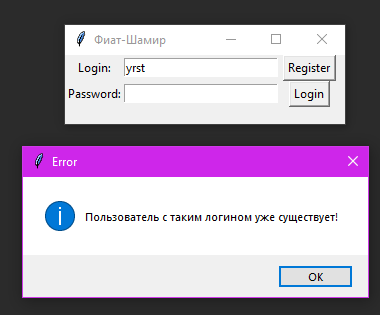
**def gen\_x()**  
Функция генерирует x на стороне клиента по формуле.

**def gen\_y()**  
Функция генерирует y на стороне клиента по формуле.

**def clicked\_log()**Функция, которая ищет пользователя в базе данных и, впоследствии, начинает вызов аккредитации данного пользователя.

Программа начинается с генерации пароля для пользователя. Далее, пользователь может попытаться “войти” на сервер, посредством ввода своего логина и пароля. При нажатии кнопки Login начинается процесс аккредитации пользователя при помощи вызова функции start у класса Server. В случае успешного происхождения появляется уведомление об успешном входе, иначе – аналогичное уведомление о неправильном логине/пароле.

**Результат работы программы**



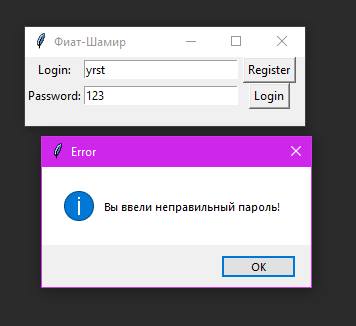


Рис.1,2 Попытки входа с неправильными данным

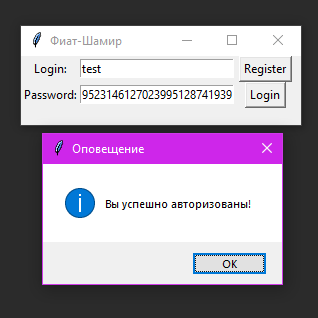


Рис.3 Вход с правильными данными

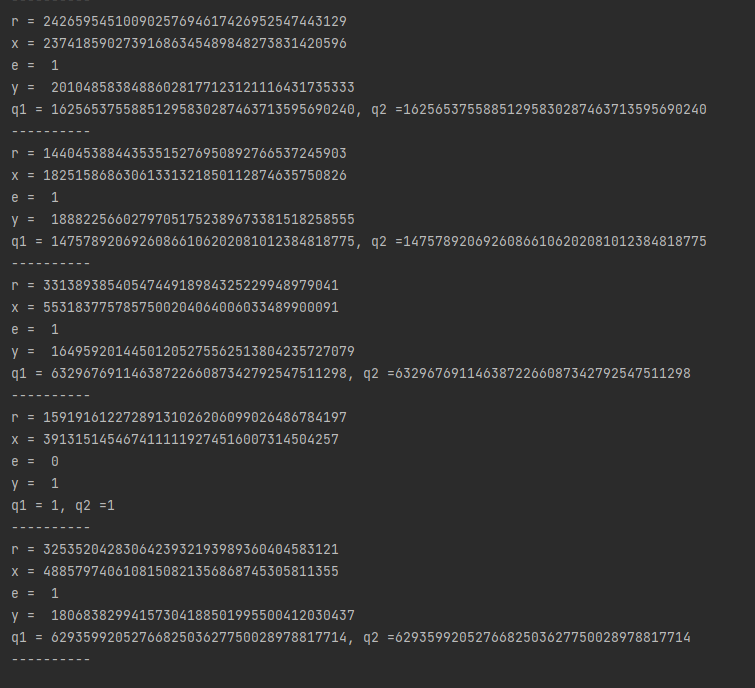


Рис.4 Этапы аккредитации

Рис.4 Первый вопрос Алисе от Боба

Рис.5 Второй вопрос Алисе

**Листинг кода**

iimport random

import json

from tkinter import \*

from tkinter import messagebox

N = 0

Keys = {}

user\_n = {}

window = Tk()

window.title("Фиат-Шамир")

window.geometry('280x70')

try:

with open("test.json", "r") as json\_file:

json\_data = json.load(json\_file)

print(json\_data)

for p in json\_data['login']:

Keys[p['user']] = p['v']

user\_n[p['user']] = p['n']

print(Keys)

except FileNotFoundError:

messagebox.showinfo("Error", 'Не удалось открыть файл')

def generate\_coprime(p):

result = random.randint(2, p)

# print(result)

while gcd(p, result) != 1:

result = random.randint(2, p)

# print(result)

return result

def gcd(a, b):

while b != 0:

r = a % b

a = b

b = r

return a

def pow\_module(a, x, p):

result = 1

a = a % p

if a == 0:

return 0;

while x > 0:

if x & 1 == 1: # если крайний правый бит степени равен lib

result = (result \* a) % p

a = (a \*\* 2) % p

x >>= 1 # побитово смещаем степень

return result

def check\_prime(p):

if p <= 1:

return False

elif p == 2:

return True

a = random.randint(2, p - 1)

# print(p, "-", a)

if pow\_module(a, (p - 1), p) != 1 or gcd(p, a) > 1:

return False

return True

def generate\_prime(left, right):

while True:

p = random.randint(left, right)

# print("--", p)

if check\_prime(p):

return p

def gen():

global N

while P := random.getrandbits(64):

if check\_prime(P):

break

while Q := random.getrandbits(64):

if check\_prime(Q):

break

print(P, Q)

N = P \* Q

print(N)

def register(name, key):

global Keys, json\_data, N

Keys[name] = key

user\_n[name] = N

json\_data['login'].append({

"user": name,

"v": key,

"n": N

})

with open('test.json', 'w') as outfile:

json.dump(json\_data, outfile, indent=4)

class Server:

def \_\_init\_\_(self, t):

self.t = t

def start(self, client):

global Keys

local\_user\_n = user\_n[client.name]

v = Keys[client.name]

for i in range(self.t):

x = client.gen\_x()

print("x =", x)

e = random.randint(0, 1)

print("e = ", e)

y = client.gen\_y(e)

print("y = ", y)

buf = x \* v

q2 = pow\_module(buf, e, local\_user\_n)

q1 = pow\_module(y, 2, local\_user\_n)

print("q1 = {}, q2 ={}".format(q1, q2))

if q1 != q2:

messagebox.showinfo("Error", 'Вы ввели неправильный пароль!')

print("Авторизация не удалась")

return

else:

print("----------")

messagebox.showinfo("Оповещение", 'Вы успешно авторизованы!')

class Client:

def \_\_init\_\_(self, name):

global N

self.name = name

self.s = generate\_coprime(N - 1)

print("s", self.s)

txt\_pass.insert(0, self.s)

v = pow\_module(self.s, 2, N)

register(self.name, v)

class Login:

def \_\_init\_\_(self, name, s, local\_user\_n):

self.name = name

self.s = s

self.local\_user\_n = local\_user\_n

def gen\_x(self):

self.r = generate\_prime(1, self.local\_user\_n - 1)

print("r =", self.r)

x = pow\_module(self.r, 2, self.local\_user\_n)

return x

def gen\_y(self, e):

d = self.r \* self.s

y = pow\_module(d, e, self.local\_user\_n)

return y

def clicked():

global json\_data

buf = str(txt.get())

for p in json\_data['login']:

if buf in p['user']:

messagebox.showinfo("Error", 'Пользователь с таким логином уже существует!')

return

Client(buf)

def clicked\_log():

global json\_data

buf = str(txt.get())

exist = False

for p in json\_data['login']:

if buf in p['user']:

local\_user\_n = p['n']

exist = True

if not exist:

messagebox.showinfo("Error", 'Пользователя с таким логином не существует!')

return

try:

pwd = int(txt\_pass.get())

except:

messagebox.showinfo("Error", 'Введите пароль!')

return

print(pwd)

user = Login(buf, pwd, local\_user\_n)

srv.start(user)

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

gen()

srv = Server(20)

lbl = Label(window, text="Login:")

lbl.grid(column=0, row=0)

txt = Entry(window, width=25)

txt.grid(column=1, row=0)

btn = Button(window, text="Register", command=clicked)

btn.grid(column=2, row=0, padx=5)

lbl = Label(window, text="Password:")

lbl.grid(column=0, row=1)

txt\_pass = Entry(window, width=25)

txt\_pass.grid(column=1, row=1)

btn\_log = Button(window, text="Login", command=clicked\_log)

btn\_log.grid(column=2, row=1, padx=5)

window.mainloop()

print("================================EVA=============================")

print(Keys)