ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Расчётно-графическое задание

по дисциплине «Защита информации»

на тему «Доказательство с нулевым знанием».

|  |
| --- |
| Выполнил:  студент гр. ИП-715  Допчук В. К.  Проверил:  ассистент  кафедры ПМиК  Петухова Я. В. |

Новосибирск, 2020г

Содержание

[Постановка задачи 3](#_Toc58421331)

[Теория 3](#_Toc58421332)

[Доказательство с нулевым знанием 3](#_Toc58421333)

[Описание реализации 5](#_Toc58421334)

[Класс Клиент 5](#_Toc58421335)

[Класс Сервер 6](#_Toc58421336)

[Основной класс Авторизации 7](#_Toc58421337)

[Описание структуры файлов 8](#_Toc58421338)

[Примеры исполнения программы 9](#_Toc58421339)

[Листинг программы 12](#_Toc58421340)

[Список используемой литературы 22](#_Toc58421341)

# Постановка задачи

**Протокол Фиата-Шамира** Для выполнения этого варианта задания необходимо разработать клиентсерверное 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.

Программа должна наглядно демонстрировать работу алгоритма, возможно (но не обязательно) в графическом режиме. Текст программы должен содержать исчерпывающие комментарии, тем не менее, следует воздержаться от описания очевидных действий. К РГР необходимо представить отчѐт, оформленный в соответствии с требованиями, предъявляемыми к работам подобного типа. В отчѐт должны быть включены в обязательном порядке: Титульный лист, содержание, постановка задачи, исходный код с комментариями, скриншоты с результатами работы программы.

# Теория

## Доказательство с нулевым знанием

Доказательство с нулевым знанием – представляет собой криптографический протокол, который позволяет одной из сторон (проверяющему Бобу) убедиться в том, что вторая сторона (доказывающая Aлиса) знает какое-либо утверждение, при этом проверяющий не получает никакой другой информации о самом утверждении. Другими словами, Алиса доказывает знание секрета, не разглашая самого секрета.

Любой протокол, реализующий доказательство с нулевым знанием, должен обладать 3 свойствами:

1. Полнота. Если доказывающий знает утверждение, то он сможет убедить в этом проверяющего.

2. Корректность. Если доказывающий не знает утверждение, то он может обмануть проверяющего только с пренебрежимо малой вероятности.

3. Нулевое разглашение. Проверяющий, даже если он ведет себя нечестно, не узнает ничего кроме самого факта, что утверждение известно доказывающему.

Протокол Фиата-Шамира

Протокол аутентификации Фиата-Шамира имеет следующие закрытые параметры: простые числа p, q, где |p|, |q| ≥ 512. Открытый параметр: n := pq. Секретным значением является число , для проверки знания s используется число . Протокол состоит из d раундов, имеющих вид

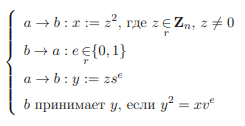


Рис 1. Один раунд

Если a хочет пройти аутентификацию, не зная s, то для каждого значения e ∈ {0, 1}, которое ему пришлёт b, a должен уметь вычислить значение ye, которое он пошлет b в качестве y. Из условия принятия y следует, что . Нетрудно видеть, что возможность  за полиномиальное время вычислить числа  с описанным выше свойством равносильна возможности за полиномиальное время вычислить . Однако можно доказать, что при описанных выше условиях на n задача вычисления  без знания p и q является вычислительно сложной. Если a не знает s, то для прохождения аутентификации он может попытаться смошенничать, аналогично тому как это делается в протоколе Шаума: a случайно выбирает число e ∈ {0, 1}, в качестве x посылает значение , и в качестве y – значение z. b примет этот ответ, если будет верно равенство , которое равносильно равенству e = e, и которое будет верно с вероятностью 1/2.

**Шаг 1.** Пользователь, узнав доступную переменную n, отправляет  . И сервер регистрирует V в качестве открытого ключа.

**Шаг 2.** **А** выбирает случайное z , такое, что  z ∈ [1, n - 1] и отсылает

x = z^2 mod n стороне **B**.

**Шаг 3.** B случайно выбирает бит e (e=0 или е=1) и отсылает его **A**

**Шаг 4.** **А** вычисляет **у** и отправляет его обратно к **B**. Если e = 0, то  y = z, иначе y = z \* s mod n.

**Шаг 5.** Если y = 0, то **B** отвергает доказательство или, другими словами, **А** не удалось доказать знание s.

В противном случае, сторона **B** проверяет, действительно ли  y^2 = x \* v ^ e mod n и, если это так, то происходит переход к **Шагу 2.**

Чтобы снизить вероятность жульничества (она равна {\displaystyle 1/2^{t})}1/2^d) **d** выбирают достаточно большим (**d**=20, **d**=40). Таким образом, **B** удостоверяется в знании **А** тогда и только тогда, когда все **d** раундов прошли успешно.

# Описание реализации

## Класс Клиент

Методы класса Client.cs

private BigInteger n { get; set; } - Число общедоступное n = p\*q

private BigInteger s { get; set; } – Секретный ключ

private BigInteger v { get; set; } – Открытый ключ

private BigInteger r { get; set; } – Рандомное значение для отправки

private BigInteger y { get; set; } – Вычисляемое на последнем этапе раунда

public Client() – Конструктор клиента, который считываетиз публичного файла значение N.

Метода, если Пользователь выбрал Регистрацию:

public void connect(BigInteger n) – Сервер передает n.

public void savePublicData() – Сохраняет для себя значение n. Чтобы потом проверяет, не изменил ли сервер его.

public bool checkNaboutServer(Server server) – Проверяет, не изменилось ли значение N у сервера, и не надо ли придумать пользователю новый пароль.

Метода, если Пользователь выбрал Авторизацию:

public void generateV(string password) – Генерируем наше значение V, используя ввод пароля с формы сайта. V = s^2 mod n.

Метода, если Пользователь выбрал Регистрацию:

public void generateSV() – Генерирует значение s по n. И V по s.

public BigInteger sendX() – Передает серверу значение х, которое считается из значение r, x = r^2 mod n

public void responseE(int e) – Получает от сервера значение числа е{0, 1}.

public BigInteger sendY() – Передает у, с учетом проверки числа е. Если 0, то у = r, если е = 1, то у = y = z \* s mod n.

## Класс Сервер

Методы класса Server.cs:

private int bitLength = 512; - количество бит для простых чисел

private BigInteger p { get; set; } – простое число

private BigInteger q { get; set; } – простое число

private BigInteger n { get; set; } – произведение простых

private BigInteger v { get; set; }- пользовательский открытый ключ

private int t { get; set; } – время(количество раундов)

private BigInteger x { get; set; } – полученный х от пользователя

private int e { get; set; } – передаваемое число 0 или 1

public Server() – Считывает значения n, p, q из приватного файла, и публикует n в публичном файле. Если файл был пуст, то сервер генерирует эти значения по-новой. И снова публикует значение n.

public bool checkLogin(string login) – Проверяет в своем файле, есть ли Пользователь с таким логином, если да. То продолжаем проверку авторизации.

public bool connect(string login, BigInteger v) – Пользователь посылает, свой логин и открытый ключ. Если Ключи совпали, то продолжается проверка, если нет, то связь будет потеряна, и Программа скажет, что Неверный логин или пароль.

public void setT(int t) – можно изменить количество раундом, поумолчанию стоит 10. По выбору стоит 40.

public void responseX(BigInteger x) – Получает от клиента значение х и записывает его себе.

public int sendE() – Передает рандомное значение е {0, 1}.

public bool responseY(BigInteger y) – Получает от клиента значние у. И проверяет его. Если всё верно, то сервер переходит к следующему раунду, если нет, то завершает проверку Авторизации и выдает «Ошибка».

Метода, если Пользователь выбрал Регистрацию:

public void saveClient(string login, BigInteger v) – Если регистрация прошла успешно, то сервер запоминает в закрытый файл логин и открытый ключ.

public void deleteClient(string login) – Если Сервер перезаписывает пользователя, то он сначала, его удаляет по его логину.

## Основной класс Авторизации

Методы класса Authorization.cs:

protected string login; - Логин из формы

protected string password; - Пароль из формы

public void auth() – Если пользователь выбрал Авторизоваться, то сервер будет проверять логин в своем файле, если не найдется – подскажет пользователю пройти Регистрацию.

public void identify(Client client, Server server) – Подключается клиент к серверу, и производится t итераций для проверки Авторизации.

public void regist() – Если пользователь выбрал Регистрацию, то сервер проверит логин в файле, если он есть, то он подскажет пройти Авторизацию.

public void register(Client client, Server server) - Подключается клиент к серверу, и производится t итераций для проверки Регистрации.

public void registerReload(Client client, Server server) – Если Сервер поменял публичное N. То перезапишем пароль для выбранного пользователя.

## Описание структуры файлов

Файл сервера со списком клиентов:

Строки в которыйх лежат Логин Пробел Открытый ключ.

Файл сервера с публичным N:

Строка с одним значением N.

Файл сервера с закрыми данными:

Строка N.

Строка p.

Строка q.

Файл клиента с закрыми данными:

Строки из Логин и Пароль.

# Примеры исполнения программы

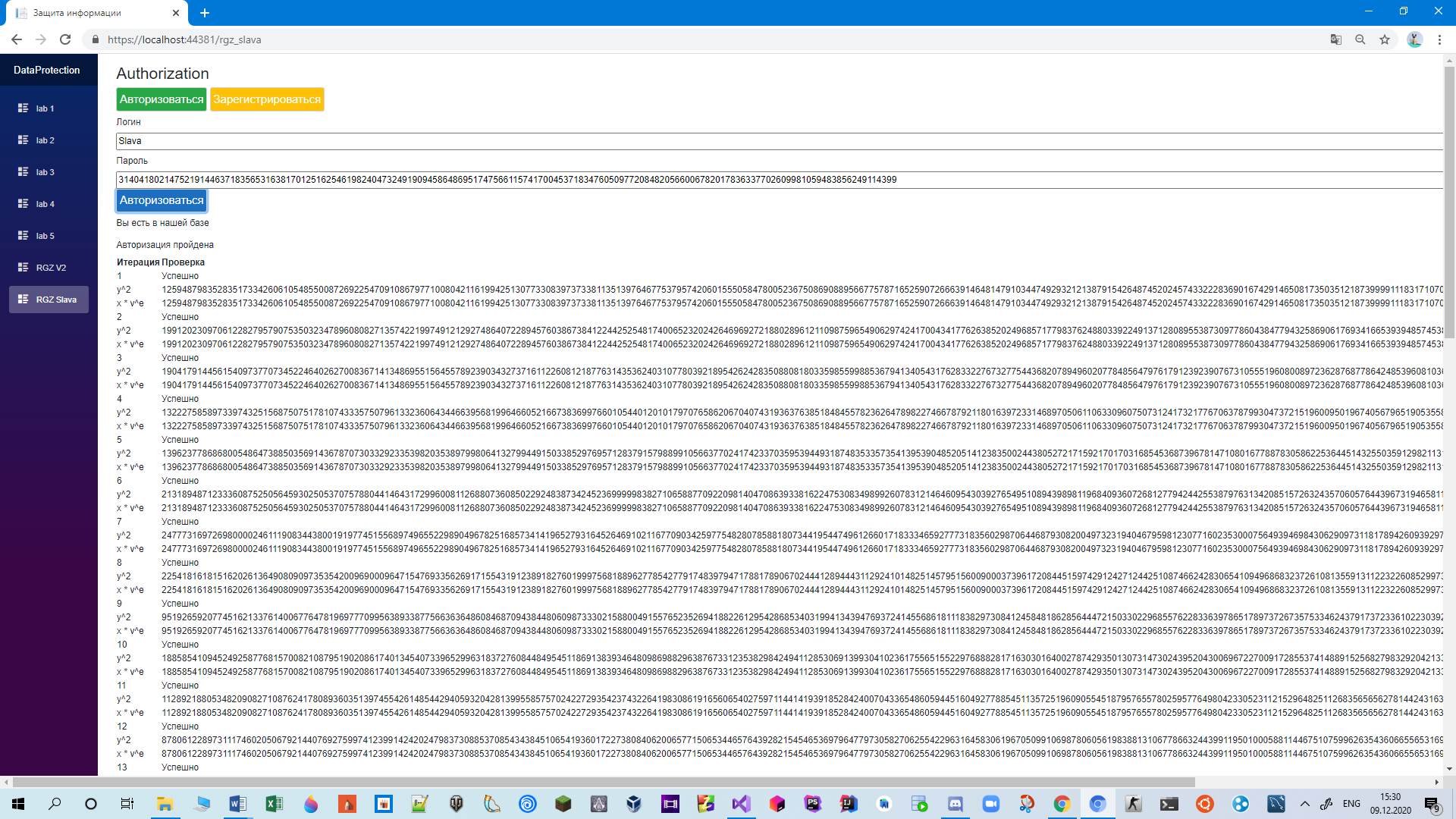


Рис 2. Пользователь успешно авторизовался. Все 40 раундов прошли успешно.

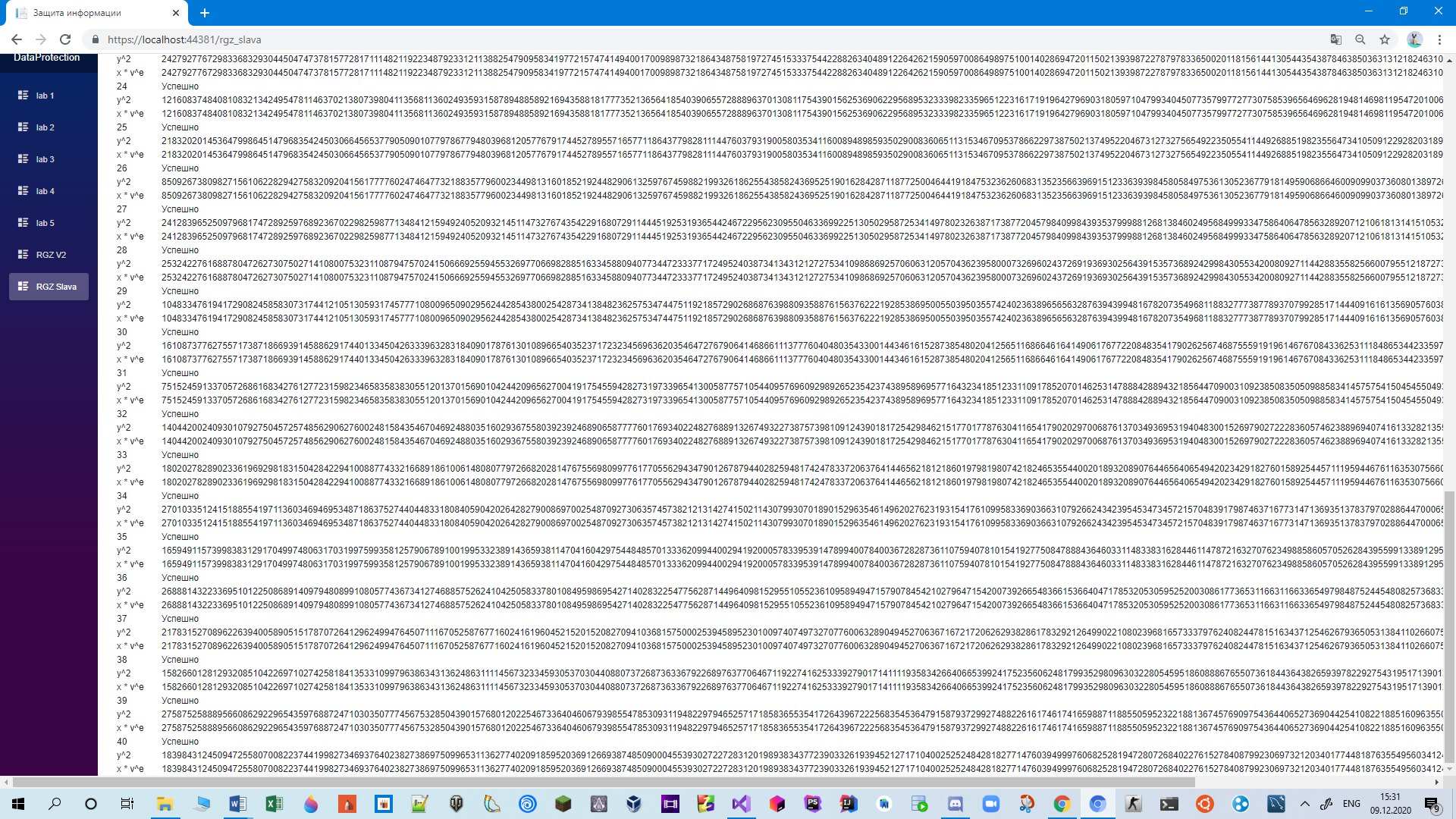


Рис 3. Продолжение. Все 40 раундов прошли успешно.

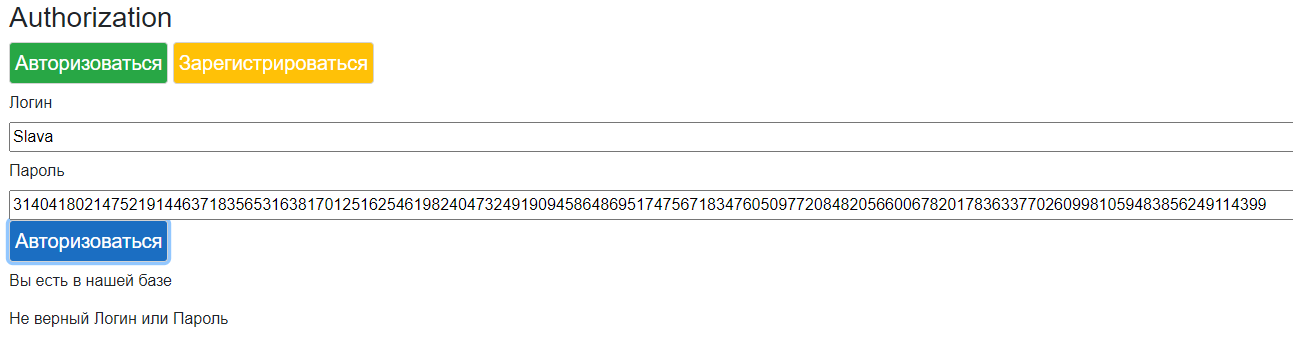


Рис 4. Пользователь ввел неверный Пароль или Логин.

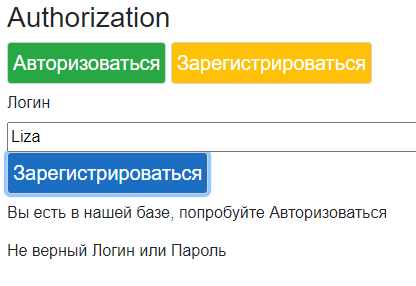


Рис 5. Пользователь уже есть в файле сервера, он просит пользователя перейти на Авторизацию.

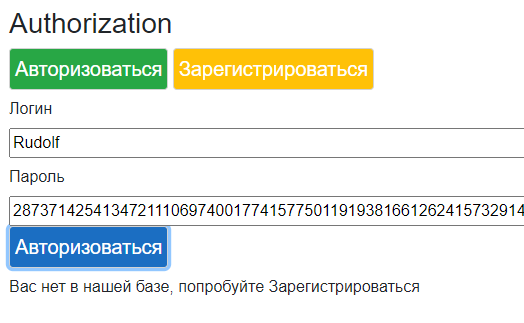


Рис 6. Пользователя нет в файле сервера.

# Листинг программы

Authorization.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using Org.BouncyCastle.Math;

namespace DataProtection.Pages.RGZ\_Slava

{

public partial class Authorization

{

/\*

\* Varibale for Output to page

\*/

public BigInteger x;

public BigInteger y;

public BigInteger v;

public int e;

/\*

\* Varibale basic output to page

\*/

protected BigInteger output;

protected List<string> outputString = new List<string>();

protected string outputMessage;

protected string outputStatus;

protected List<string> outputCheckLeft = new List<string>();

protected List<string> outputCheckRight = new List<string>();

/\*

\* Varibale login and password

\*/

protected string login;

protected string password;

/\*

\* Varibale for page

\*/

protected int choose = 0;

protected bool tableShow = false;

public void regist()

{

Server server = new Server();

Client client = new Client();

if (!string.IsNullOrEmpty(login)) {

client.saveLogin(login);

if (client.checkNaboutServer(server)) {

if (server.checkLogin(client.getLogin())) {

outputStatus = "Вы есть в нашей базе, попробуйте Авторизоваться";

} else {

outputStatus = "Вас нет в нашей базе, генерируем вам пароль";

tableShow = false;

outputMessage = "";

register(client, server);

}

} else {

outputStatus = "Сменился сервер, генерируем вам пароль";

tableShow = false;

outputMessage = "";

registerReload(client, server);

}

outputMessage = "";

}

}

public void auth()

{

Server server = new Server();

Client client = new Client();

if (!string.IsNullOrEmpty(login)) {

client.saveLogin(login);

if (client.checkNaboutServer(server)) {

if (server.checkLogin(client.getLogin())) {

outputStatus = "Вы есть в нашей базе";

identify(client, server);

} else {

outputStatus = "Вас нет в нашей базе, попробуйте Зарегистрироваться";

tableShow = false;

outputMessage = "";

}

} else {

outputStatus = "Сменился сервер, генерируем вам пароль";

tableShow = false;

outputMessage = "";

registerReload(client, server);

}

}

}

public void register(Client client, Server server)

{

client.connect(server.getN());

client.generateSV();

server.connect(client.getV());

server.savePublicData();

client.savePublicData();

outputString = new List<string>();

server.setT(40);

for (int i = 0; i < server.getT(); i++) {

server.responseX(client.sendX());

client.responseE(server.sendE());

if (server.responseY(client.sendY())) {

outputString.Add("Успешно");

outputCheckLeft.Add(server.getLeft());

outputCheckRight.Add(server.getRight());

} else {

outputString.Add("Ошибка");

outputCheckLeft.Add(server.getLeft());

outputCheckRight.Add(server.getRight());

break;

}

}

if (outputString.Find(x => x == "Ошибка") == "Ошибка") {

outputMessage = "Регистрация не корректна";

} else {

outputMessage = "Регистрация пройдена";

client.savePassword();

server.saveClient(client.getLogin(), client.getV());

tableShow = true;

}

}

public void identify(Client client, Server server)

{

outputString = new List<string>();

client.generateV(password);

if (server.connect(client.getLogin(), client.getV())) {

server.setT(40);

for (int i = 0; i < server.getT(); i++) {

server.responseX(client.sendX());

client.responseE(server.sendE());

if (server.responseY(client.sendY())) {

outputString.Add("Успешно");

outputCheckLeft.Add(server.getLeft());

outputCheckRight.Add(server.getRight());

} else {

outputString.Add("Ошибка");

outputCheckLeft.Add(server.getLeft());

outputCheckRight.Add(server.getRight());

break;

}

}

if (outputString.Find(x => x == "Ошибка") == "Ошибка") {

outputMessage = "Авторизация не корректна";

} else {

outputMessage = "Авторизация пройдена";

tableShow = true;

}

} else {

outputMessage = "Не верный Логин или Пароль";

}

}

public void registerReload(Client client, Server server)

{

client.connect(server.getN());

client.generateSV();

server.connect(client.getV());

server.savePublicData();

client.savePublicData();

outputString = new List<string>();

server.setT(40);

for (int i = 0; i < server.getT(); i++) {

server.responseX(client.sendX());

client.responseE(server.sendE());

if (server.responseY(client.sendY())) {

outputString.Add("Успешно");

} else {

outputString.Add("Ошибка");

break;

}

}

if (outputString.Find(x => x == "Ошибка") == "Ошибка") {

outputMessage = "Регистрация не корректна";

} else {

outputMessage = "Регистрация пройдена";

client.deletePassword();

client.savePassword();

server.deleteClient(client.getLogin());

server.saveClient(client.getLogin(), client.getV());

tableShow = true;

}

}

}

}

Client.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Org.BouncyCastle.Math;

using DataProtection.Engine.Managers;

using System.IO;

namespace DataProtection.Pages.RGZ\_Slava

{

public class Client

{

private int bitLength = 510;

private BigInteger n { get; set; }

private BigInteger s { get; set; }

private BigInteger v { get; set; }

private BigInteger r { get; set; }

private BigInteger y { get; set; }

private int e { get; set; }

private string login { get; set; }

private string pathDataPublicServer = "Resource\\RGZ\_Slava\\publicData.txt";

private string pathDataPublicClient = "Resource\\RGZ\_Slava\\publicDataClient.txt";

private string pathPasswords = "Resource\\RGZ\_Slava\\clientPasswords.txt";

public Client(string login, string password, Server server) {

}

public Client()

{

read(pathDataPublicClient);

}

public void saveLogin(string login)

{

this.login = login;

}

public void generateV(string password)

{

if (password == null || password == "" || !checkOnSymbol(password)) {

s = new BigInteger("2");

} else {

s = new BigInteger(password);

}

v = MyModPowBigInteger.FastModuloExponentiation(s, BigInteger.Two, n);

}

private bool checkOnSymbol(string password)

{

for (int i = 0; i < password.Length; i++) {

if (password[i] > '9' || password[i] < '0') {

return false;

}

}

return true;

}

public string getLogin()

{

return login;

}

public void connect(BigInteger n)

{

this.n = n;

}

public void generateSV()

{

EvklidBigInteger evklid = new EvklidBigInteger();

BigInteger tmp\_s;

do {

tmp\_s = BigInteger.ProbablePrime(bitLength, new Random());

} while (!tmp\_s.IsProbablePrime(10) && evklid.gcd(tmp\_s, n).CompareTo(BigInteger.One) != 0);

s = tmp\_s;

v = MyModPowBigInteger.FastModuloExponentiation(s, BigInteger.Two, n);

}

public void savePublicData()

{

using (StreamWriter writer = new StreamWriter(pathDataPublicClient)) {

writer.Write(n.ToString());

}

}

public bool read(string path)

{

string clientData;

using (StreamReader reader = new StreamReader(path)) {

clientData = reader.ReadLine();

}

if (clientData != null && !clientData.Equals("")) {

n = new BigInteger(clientData);

return true;

}

return false;

}

public bool checkNaboutServer(Server server)

{

string clientData, serverData;

using (StreamReader reader = new StreamReader(pathDataPublicClient)) {

clientData = reader.ReadLine();

}

using (StreamReader reader = new StreamReader(pathDataPublicServer)) {

serverData = reader.ReadLine();

}

if (clientData != null && serverData != null && !clientData.Equals("") && !serverData.Equals("") && clientData.Equals(serverData)) {

n = new BigInteger(clientData);

return true;

}

return false;

}

public void savePassword()

{

using (StreamWriter writer = new StreamWriter(pathPasswords, true)) {

writer.Write(login + " " + s.ToString() + "\n");

}

}

private void generateR()

{

r = BigInteger.ProbablePrime(bitLength - 1, new Random());

}

public BigInteger sendX()

{

generateR();

return MyModPowBigInteger.FastModuloExponentiation(r, BigInteger.Two, n);

}

public void responseE(int e)

{

this.e = e;

}

public BigInteger sendY()

{

if (e == 0) {

return r;

} else if (e == 1) {

return MyModPowBigInteger.FastModuloExponentiation(

r.Multiply(MyModPowBigInteger.FastModuloExponentiation(s, BigInteger.One, n)),

BigInteger.One,

n);

}

return BigInteger.Zero;

}

private void setN(BigInteger n)

{

this.n = n;

}

public BigInteger getV()

{

return v;

}

public BigInteger getS()

{

return s;

}

public void deletePassword()

{

List<string> allClient = new List<string>();

using (StreamReader reader = new StreamReader(pathPasswords)) {

string line;

while ((line = reader.ReadLine()) != null) {

string[] split = line.Split(" ");

if (!split[0].Equals(login)) {

allClient.Add(line);

}

}

}

using (StreamWriter writer = new StreamWriter(pathPasswords)) {

foreach (var iter in allClient) {

writer.Write(iter + "\n");

}

}

}

}

}

Server.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Org.BouncyCastle.Math;

using DataProtection.Engine.Managers;

using System.IO;

namespace DataProtection.Pages.RGZ\_Slava

{

public class Server

{

static int check = 0;

private int bitLength = 512;

private BigInteger p { get; set; }

private BigInteger q { get; set; }

private BigInteger g { get; set; }

private BigInteger n { get; set; }

private BigInteger v { get; set; }

private int t { get; set; }

private BigInteger x { get; set; }

private int e { get; set; }

private bool ControlCheck { get; set; }

private string left\_check;

private string right\_check;

private string path = "Resource\\RGZ\_Slava\\serverData.txt";

private string pathList = "Resource\\RGZ\_Slava\\serverList.txt";

string pathDataPublicServer = "Resource\\RGZ\_Slava\\publicData.txt";

public Server()

{

if (read(path)) {

} else {

generate();

saveData();

savePublicData();

}

}

public bool checkLogin(string login)

{

using (StreamReader reader = new StreamReader(pathList)) {

string line;

while ((line = reader.ReadLine()) != null) {

string[] split = line.Split(" ");

if (split[0].Equals(login)) {

return true;

}

}

}

return false;

}

public void savePublicData()

{

using (StreamWriter writer = new StreamWriter(pathDataPublicServer)) {

writer.Write(n.ToString());

}

}

private void saveData()

{

using (StreamWriter writer = new StreamWriter(path)) {

writer.Write(n.ToString() + "\n");

writer.Write(p.ToString() + "\n");

writer.Write(q.ToString());

}

}

bool read(string path)

{

using (StreamReader reader = new StreamReader(path)) {

string line;

List<BigInteger> numbers = new List<BigInteger>();

int i = 0;

while ((line = reader.ReadLine()) != null) {

numbers.Add(new BigInteger(line));

i++;

}

if (i == 3) {

n = numbers[0];

p = numbers[1];

q = numbers[2];

return true;

} else {

return false;

}

}

}

public void saveClient(string login, BigInteger v)

{

using (StreamWriter writer = new StreamWriter(pathList, true)) {

writer.Write(login + " " + v.ToString() + "\n");

}

}

/\*

\* Generate P and Q

\*/

private void generate()

{

EvklidBigInteger evklid = new EvklidBigInteger();

BigInteger tmp\_p, tmp\_q;

do {

do {

tmp\_p = BigInteger.ProbablePrime(bitLength, new Random());

tmp\_q = BigInteger.ProbablePrime(bitLength, new Random());

} while (!tmp\_p.IsProbablePrime(10) && !tmp\_q.IsProbablePrime(10));

} while (evklid.gcd(tmp\_q, tmp\_p).CompareTo(BigInteger.One) != 0);

p = tmp\_p;

q = tmp\_q;

n = q.Multiply(p);

t = 10;

}

public bool connect(string login, BigInteger v)

{

using (StreamReader reader = new StreamReader(pathList)) {

string line;

while ((line = reader.ReadLine()) != null) {

string[] split = line.Split(" ");

if (split[0].Equals(login)) {

if (split[1].Equals(v.ToString())) {

this.v = v;

return true;

}

}

}

}

return false;

}

public void connect(BigInteger v)

{

this.v = v;

}

public void responseX(BigInteger x)

{

this.x = x;

}

public int sendE()

{

e = new Random().Next(0, 1);

return e;

}

public bool responseY(BigInteger y)

{

if (y.CompareTo(BigInteger.Zero) == 0) {

check++;

return false;

}

BigInteger left = y.Multiply(y);

BigInteger right = MyModPowBigInteger.FastModuloExponentiation(

x.Multiply(MyModPowBigInteger.FastModuloExponentiation(v, new BigInteger(e.ToString()), n)),

BigInteger.One,

n);

left\_check = left.ToString();

right\_check = right.ToString();

if (left.CompareTo(right) == 0) {

return true;

}

return false;

}

public void setV(BigInteger v)

{

this.v = v;

}

public BigInteger getV()

{

return v;

}

public BigInteger getN()

{

return n;

}

public void setT(int t)

{

this.t = t;

}

public void deleteClient(string login)

{

List<string> allClient = new List<string>();

using (StreamReader reader = new StreamReader(pathList)) {

string line;

while ((line = reader.ReadLine()) != null) {

string[] split = line.Split(" ");

if (!split[0].Equals(login)) {

allClient.Add(line);

}

}

}

using (StreamWriter writer = new StreamWriter(pathList)) {

foreach (var iter in allClient) {

writer.Write(iter + "\n");

}

}

}

public int getT()

{

return t;

}

public string getLeft()

{

return left\_check;

}

public string getRight()

{

return right\_check;

}

}

}

# Список используемой литературы

1. Рябко Б. Я., Фионов А. Н. Р 98 Криптографические методы защиты информации: Учебное пособие для вузов. – 2-е издание, стереотип. – М.: Горячая линия–Телеком, 2012. – 229 c.: ил. ISBN 978-5-9912-0286-2.
2. Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А. В. Основы криптографии. М.: Гелиос АРВ, 2001. 479 с.
3. Протоколы безопасности [Электронный ресурс] (08.12.2020)

URL: http://intsysjournal.org/pdfs/21-3/65-105-Mironov.pdf