

Криптографические протоколы
Лабораторная работа № 3
«Протоколы распределения ключей»
Отчет по лабораторной работе:

8. Приложение с графическим интерфейсом, реализующее крипто-протокол «Взаимоблокировка» - («держась за руки»).

Алгоритм шифрования с открытым ключом - схема Эль-Гамала.

Описание функций программы:

Main:

Из интересного:

13 – ссылка на fhtml файл с формой программы

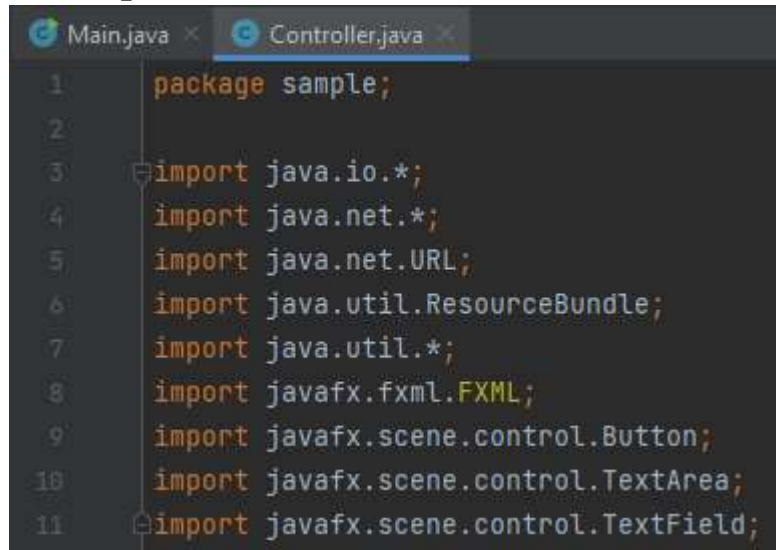
15 – размеры программы

16 – запретить растягивать окно программы

```
App > src > sample > Main
Main.java x Controller.java x
1 package sample;
2
3 import javafx.application.Application;
4 import javafx.fxml.FXMLLoader;
5 import javafx.scene.Parent;
6 import javafx.scene.Scene;
7 import javafx.stage.Stage;
8
9 public class Main extends Application {
10
11     @Override
12     public void start(Stage primaryStage) throws Exception{
13         Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
14         primaryStage.setTitle("Messenger");
15         primaryStage.setScene(new Scene(root, 550, 450));
16         primaryStage.setResizable(false);
17         primaryStage.show();
18     }
19
20
21     public static void main(String[] args) {
22         launch(args);
23     }
24 }
```

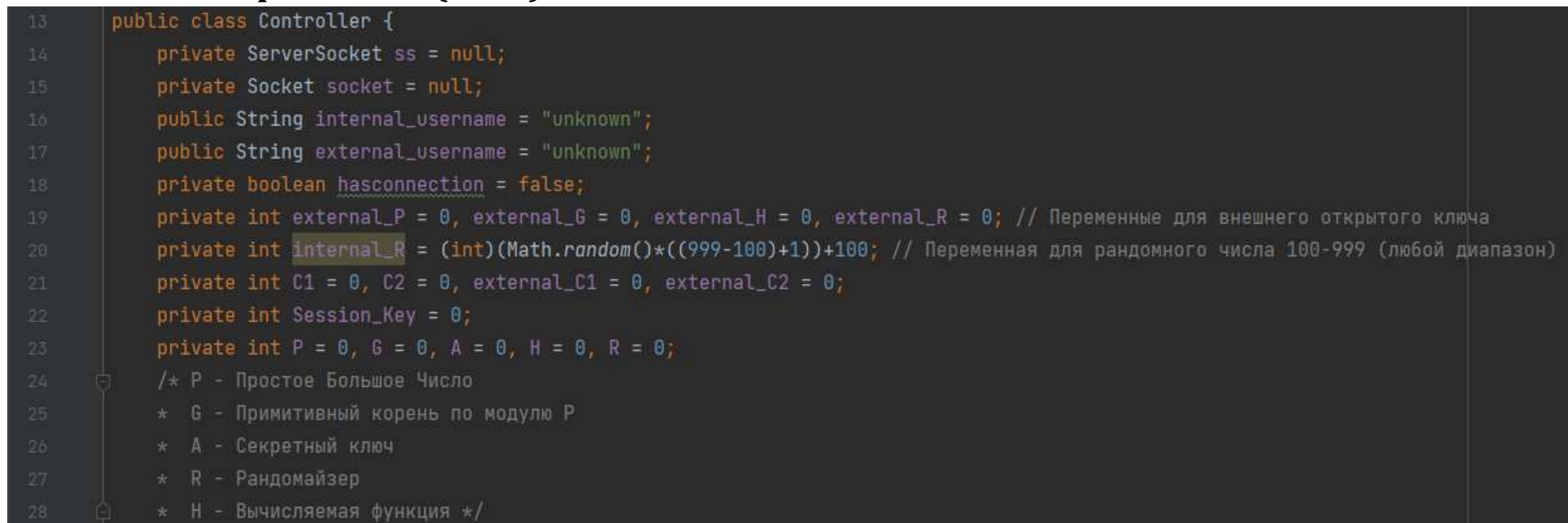
2. Controller – основной класс разбитый на блоки:

Импорт:



```
1 package sample;
2
3 import java.io.*;
4 import java.net.*;
5 import java.net.URL;
6 import java.util.ResourceBundle;
7 import java.util.*;
8 import javafx.fxml.FXML;
9 import javafx.scene.control.Button;
10 import javafx.scene.control.TextArea;
11 import javafx.scene.control.TextField;
```

Глобальные переменные (поля):



```
13 public class Controller {
14     private ServerSocket ss = null;
15     private Socket socket = null;
16     public String internal_username = "unknown";
17     public String external_username = "unknown";
18     private boolean hasconnection = false;
19     private int external_P = 0, external_G = 0, external_H = 0, external_R = 0; // Переменные для внешнего открытого ключа
20     private int internal_R = (int)(Math.random()*((999-100)+1))+100; // Переменная для случайного числа 100-999 (любой диапазон)
21     private int C1 = 0, C2 = 0, external_C1 = 0, external_C2 = 0;
22     private int Session_Key = 0;
23     private int P = 0, G = 0, A = 0, H = 0, R = 0;
24     /* P - Простое Большое Число
25     * G - Простой корень по модулю P
26     * A - Секретный ключ
27     * R - Рандомайзер
28     * H - Вычисляемая функция */
```

1. **Internal_username** – свой никнейм, **external_username** получаемый никнейм собеседника.
2. **Hasconnection** – флаг, отслеживающий нажатия на клавиши **Create/Connect**
3. **External_P,G,H,R** – Внешний открытый ключ и Rb/Ra в зависимости от положения сервер или клиент.
4. **C1, C2, external_C1, external_C2** – Шифротекст (свой и внешний), содержащий в себе Ra/Rb в зависимости от положения сервер или клиент.
5. **Session_Key** – Вычисляемый сессионный ключ $K = Ra \wedge Rb$

```
104 //-----  
105 @ public static String Encryption (String Text, int Key){...}  
116 @ public static String Decryption (String Text, int Key){...}  
127 public static int Gen_C1 (int external_G, int R, int external_P){...}  
131 public static int Gen_C2 (int M, int external_H, int R, int external_P){...}  
135 public int Dec_C (int C1, int C2, int P, int A){...}  
142 public static int GetPRoot(int p) {...}  
149 public static boolean IsPRoot(long p, long a) {...}  
164 public static int modInverse(int a, int m){...}  
170 public static int gcd(int a, int b) {...}  
189 private static boolean TestRabinMiller(int number, int mod){...}  
203 public static int power(int x, int y, int p) {...}  
226 public static boolean isPrime(int n, int k) {...}  
237 public static int generationLargeNumber(){...}  
245 //-----
```

Encryption – Шифрование

Decryption – Дешифрование

Gen_C1, Gen_C2 генерация **C1** и **C2** по формулам ниже

GetPRoot + isPRoot – первообразный (примитивный корень) простого числа

modInverse – обратный в кольце по модулю

gcd – НОД двух чисел. Не используется.

generationLargeNumber – генерация большого простого числа с проверкой **isPrime** на простоту и **TestRabinMiller** вероятностный полиномиальный тест простоты
power – возведение в степень по модулю

Интерфейс программы:



Функционал:

1. Поля по умолчанию **localhost** и **6996**
2. Пока не была нажата кнопка **Connect/Create** можно **поменять ник** с unknown на свой путем отправки его в чат (неограниченное кол-во раз).
3. **Connect** – кнопка для подключения к серверу (Bob -> Alice)
4. **Create** – кнопка для создания сервера (Alice)

```

52 CreateButton.setOnAction(event -> {
53     String s_port = port_field.getText().trim();
54     int port = Integer.parseInt(s_port); // (может быть любое число от 1025 до 65535)
55     System.out.println("Port in field: " + port);
56     hasconnection = true;
57     try {
58         ss = new ServerSocket(port); // создаем сокет сервера и привязываем его к вышеуказанному порту
59         chat_field.appendText(s: "\nПорт для подключения: " + port + "\nОжидание подключения...");
60         new ReadMsgServer().start(); // нить читающая сообщения из сокета в бесконечном цикле
61     } catch (Exception x) { x.printStackTrace(); }
62 });

```

Считываем значение из поля Port (по умолчанию: 6996) и выводим в консоль как log.

Меняем значение hasconnection чтобы дать понять программе что пора перестать менять ники (функционал программы описан ниже) и уже быть готовой для начальной инициализации.

Connect по аналогии.

```

63 ConnectButton.setOnAction(event -> {
64     String s_port = port_field.getText().trim();
65     int port = Integer.parseInt(s_port); // здесь обязательно нужно указать порт к которому привязывается сервер.
66     String s_ip = ip_field.getText().trim(); // это IP-адрес компьютера, где исполняется наша серверная программа.
67     //localhost ip
68     // Здесь указан адрес того самого компьютера где будет исполняться и клиент.
69     try {
70         InetAddress ipAddress = InetAddress.getByName(s_ip); // создаем объект который отображает вышеописанный IP-адрес.
71         //Конвертируем IP
72         chat_field.appendText(s: "\nПодключение к серверу с адресом: " + s_ip + ":" + port);
73         socket = new Socket(ipAddress, port); // создаем сокет используя IP-адрес и порт сервера.
74         new ReadMsgClient().start(); // нить читающая сообщения из сокета в бесконечном цикле
75         chat_field.appendText(s: "\nПодключение успешно!");
76         hasconnection = true;
77     } catch (Exception x) {
78         System.out.println("\nПодключение не удалось!");
79     }
80 });

```



```

81  SendButton.setOnAction(event -> {
82      if (hasconnection == false) {
83          internal_username = message_field.getText();
84          chat_field.appendText(s: "\nUsername has been changed to '" + internal_username + "'");
85          message_field.setText(null);
86      }
87
88      if (hasconnection == true) {
89          try {
90              OutputStream sout = socket.getOutputStream();
91              DataOutputStream out = new DataOutputStream(sout);
92              String line = message_field.getText();
93              chat_field.appendText(s: "\n[" + internal_username + "]: " + line);
94              line = Encryption(line, Session_Key);
95              out.writeUTF(line); // отсылаем клиенту строку текста.
96              message_field.setText(null); // Очищаем строку для следующего ввода
97              out.flush(); // заставляем поток закончить передачу данных.
98          } catch (IOException e) {
99              System.out.println("Проверьте подключение к сокету");
100          }
101      }
102  });

```

92 – берем текст из поля

93 – Выводим его в чат

94 – Шифруем сессионным ключом

95 – Отсылаем Шифротекст (набор символов)

96 – Очищаем поле для последующего ввода, чтобы не делать это вручную

Теоретическая справка: (какой информации придерживался при реализации)

Криптопротокол 1.3.

Взаимоблокировка («держась за руки»)

1. Алиса отправляет Бобу свой открытый ключ.
2. Боб отправляет Алисе свой открытый ключ.
3. Алиса генерирует случайное число R_A , шифрует его с помощью открытого ключа Боба. Половину зашифрованного сообщения она отправляет Бобу.
4. Боб генерирует случайное число R_B , шифрует его с помощью открытого ключа Алисы. Половину зашифрованного сообщения он отправляет Алисе.
5. Алиса отправляет Бобу оставшуюся половину зашифрованного сообщения.
6. Боб складывает две половины сообщения Алисы и расшифровывает сообщение Алисы своим закрытым ключом. Затем Боб отправляет Алисе вторую половину своего шифрованного сообщения.
7. Алиса складывает две половины сообщения Боба и расшифровывает сообщение Боба своим закрытым ключом.
8. Алиса и Боб вычисляют сеансовый ключ: $K = R_A \oplus R_B$.
9. Алиса и Боб шифруют свои сообщения, используя сеансовый ключ K .



Криптосистема Эль-Гамала: зашифрование

1

Произвольно
выбрать
рандомизатор r
– целое число из
 $[1; p-1]$

2

Открытое
сообщение
разбить на блоки
 $M < p$

3

Уравнения шифрования

$$C_1 = g^r \pmod{p}; \quad C_2 = M \cdot h^r \pmod{p}$$



$(C_1; C_2)$ – шифротекст текста M

Криптосистема Эль-Гамала : расшифрование

Уравнения расшифрования

$$M = C_2 \cdot (C_1^a)^{-1} \pmod{p}$$

Зашифрование
на открытом
ключе получателя
шифротекста



Расшифрование
на секретном
ключе получателя
шифротекста

Криптосистема Эль-Гамала : пример

Пример: сгенерировать ключи и зашифровать с помощью шифра Эль-Гамала сообщение **СКЛО**.

Решение: **СКЛО** = 21 14 15 18 = блоки 2114, 1518.

$$p=2179; g=7; a=8$$

$$h=g^a \bmod p=7^8 \bmod 2179=1346.$$

(2179, 7, 1346) – открытый ключ. Выбираем $r = 9$.

Зашифрование: **$M=2114$** .

$$C_1=g^r \bmod p=7^9 \bmod 2179=706;$$

$$C_2=M \cdot h^r \bmod p=2114 \cdot 1346^9 \bmod 2179=51.$$

Шифротекст для **$M=2114$** – пара **(706, 51)**

Криптосистема Эль-Гамала : пример

Аналогично шифротекст для **$M=1518$** – пара **(706, 1055)**.

Расшифрование:

$$M=C_2 \cdot (C_1^a)^{-1} \bmod 2179 = 51 \cdot (706^8)^{-1} \bmod 2179 =$$

$$= 51 \cdot (1139)^{-1} \bmod 2179 = 51 \cdot 2135 \bmod 2179 = 2114 = \text{СК}.$$

Расшифрование второго блока аналогично.



<https://present5.com/algorithm-el-gamalya-multiplikativnaya-gruppa-zn-gruppa-g/>

Основная функция с реализацией протокола взаимоблокировки «держась за руки» для:

1. Сервера (нажали Create)
2. Клиента (нажали Connect)

```

246 private class ReadMsgServer extends Thread {
247     @Override
248     public void run() {
249         try {
250             socket = ss.accept(); // заставляем сервер ждать подключений
251
252             //Генерация открытого ключа
253             P = generationLargeNumber();
254             chat_field.appendText(s: "\nПростое большое число P: " + P);
255             G = GetPRoot(P);
256             chat_field.appendText(s: "\nПервообразный корень G: " + G);
257             A = (int)(Math.random()*((P-2)-1)+1)+1; // (Math.random()*((max-min)+1))+min;
258             chat_field.appendText(s: "\nСекретный ключ A: " + A);
259             H = power(G,A,P); // Возведение в степень по модулю
260             chat_field.appendText(s: "\nh = g^a mod p = " + H);
261             // Генерируем рандомизатор r - целое число из [1;p-1]
262             R = (int)(Math.random()*((P-1)-1)+1);
263             chat_field.appendText(s: "\nRandomizer: " + R);
264             chat_field.appendText(s: "\nRa: " + internal_R);
265
266             chat_field.appendText(s: "\nGot a client.");
267             InputStream sin = socket.getInputStream();
268             // Конвертируем потоки в другой тип, чтоб легче обрабатывать текстовые сообщения.
269             // Легче работать со строками
270             DataInputStream in = new DataInputStream(sin);
271
272             OutputStream sout = socket.getOutputStream();
273             DataOutputStream out = new DataOutputStream(sout);
274
275             out.writeInt(P); // Передаем 3 компоненты открытого ключа сервера (Алисы)
276             out.writeInt(G);
277             out.writeInt(H);
278             out.writeUTF(internal_username); // Отправляем никнейм сервера клиенту
279             out.flush(); // заставляем поток закончить передачу данных.
280
281             external_P = in.readInt(); // Ждем пока Bob пришлет все 3 компоненты своего открытого ключа
282             external_G = in.readInt();
283             external_H = in.readInt();
284             external_username = in.readUTF(); // Считываем никнейм клиента
285
286
287             chat_field.appendText(s: "\nПришел открытый ключ: [" + external_P + ";" + external_G + ";" + external_H + "]");
288             C1 = Gen_C1(external_G, R, external_P); // Шифрование M={C1,C2}=Ra открытым ключом Bob
289             C2 = Gen_C2(internal_R, external_H, R, external_P);
290             chat_field.appendText(s: "\nШифротекст: [" + C1 + ";" + C2 + "]");
291
292             // Протокол взаимоблокировки
293             out.writeInt(C1); // Отправляем первую часть
294             external_C1 = in.readInt(); // Принимаем первую часть
295             out.writeInt(C2); // Отправляем вторую часть
296             external_C2 = in.readInt(); // Принимаем вторую часть
297             chat_field.appendText(s: "\nC1: " + external_C1 + " C2: " + external_C2);
298             external_R = Dec_C(external_C1, external_C2, P, A); // Расшифровка двух частей
299             chat_field.appendText(s: "\nRb: " + external_R);
300
301             Session_Key = internal_R ^ external_R; // K = Ra xor Rb
302             chat_field.appendText(s: "\nSession Key: " + Session_Key);
303
304             String line = null; //Создаем пустую строку "буфер"
305             while(true) {
306                 line = in.readUTF(); // ожидаем пока клиент пришлет строку текста.
307                 line = Decryption(line,Session_Key);
308                 chat_field.appendText(s: "\n[" + external_username + "]: " + line);
309             }
310         } catch (IOException e) {
311             System.out.println("Исключение ReadMsgServer");
312         }
313     }
314 }

```



```

315 private class ReadMsgClient extends Thread {
316     @Override
317     public void run() {
318         try {
319
320             //Генерация открытого ключа
321             P = generationLargeNumber();
322             chat_field.appendText(s: "\nПростое большое число P: " + P);
323             G = GetPRoot(P);
324             chat_field.appendText(s: "\nПервообразный корень G: " + G);
325             A = (int)(Math.random()*(((P-2)-1)+1))+1; // (Math.random()*((max-min)+1))+min;
326             chat_field.appendText(s: "\nСекретный ключ A: " + A);
327             H = power(G,A,P); // Возведение в степень по модулю
328             chat_field.appendText(s: "\nh = g^a mod p = " + H);
329             // Генерируем рандомизатор r - целое число из [1;p-1]
330             R = (int)(Math.random()*(((P-1)-1)+1))+1;
331             chat_field.appendText(s: "\nRandomizer: " + R);
332             chat_field.appendText(s: "\nRb: " + internal_R);
333
334             InputStream sin = socket.getInputStream();
335             // Конвертируем потоки в другой тип, чтоб легче обрабатывать текстовые сообщения.
336             // Легче работать со строками
337             DataInputStream in = new DataInputStream(sin);
338
339             OutputStream sout = socket.getOutputStream();
340             DataOutputStream out = new DataOutputStream(sout);
341
342             external_P = in.readInt(); // Ждем пока Alice пришлет все 3 компоненты своего открытого ключа
343             external_G = in.readInt();
344             external_H = in.readInt();
345             external_username = in.readUTF(); // Считываем никнейм сервера
346
347             out.writeInt(P); // Передаем 3 компоненты открытого ключа (Bob)
348             out.writeInt(G);
349             out.writeInt(H);
350             out.writeUTF(internal_username); // Отправляем никнейм клиента серверу
351             out.flush(); // заставляем поток закончить передачу данных.
352
353             chat_field.appendText(s: "\nПришел открытый ключ: [" + external_P + ";" + external_G + ";" + external_H + "]");
354             C1 = Gen_C1(external_G, R, external_P);
355             C2 = Gen_C2(internal_R, external_H, R, external_P);
356             chat_field.appendText(s: "\nШифротекст: [" + C1 + ";" + C2 + "]");
357
358             // Протокол взаимоблокировки
359             external_C1 = in.readInt(); // Принимаем первую часть
360             out.writeInt(C1); // Отправляем первую часть
361             external_C2 = in.readInt(); // Принимаем вторую часть
362             external_R = Dec_C(external_C1, external_C2, P, A); // Расшифровка двух частей
363             chat_field.appendText(s: "\nRa: " + external_R);
364             out.writeInt(C2); // Отправляем вторую часть
365
366             Session_Key = internal_R ^ external_R; // K = Ra xor Rb
367             chat_field.appendText(s: "\nSession Key: " + Session_Key);
368
369             String line = null; //Создаем пустую строку "буфер"
370             while(true) {
371                 line = in.readUTF(); // ожидаем пока клиент пришлет строку текста.
372                 line = Decryption(line, Session_Key);
373                 chat_field.appendText(s: "\n[" + external_username + "]: " + line);
374             }
375         } catch (IOException e) {
376             System.out.println("Исключение ");
377         }
378     }
379 }
380 }

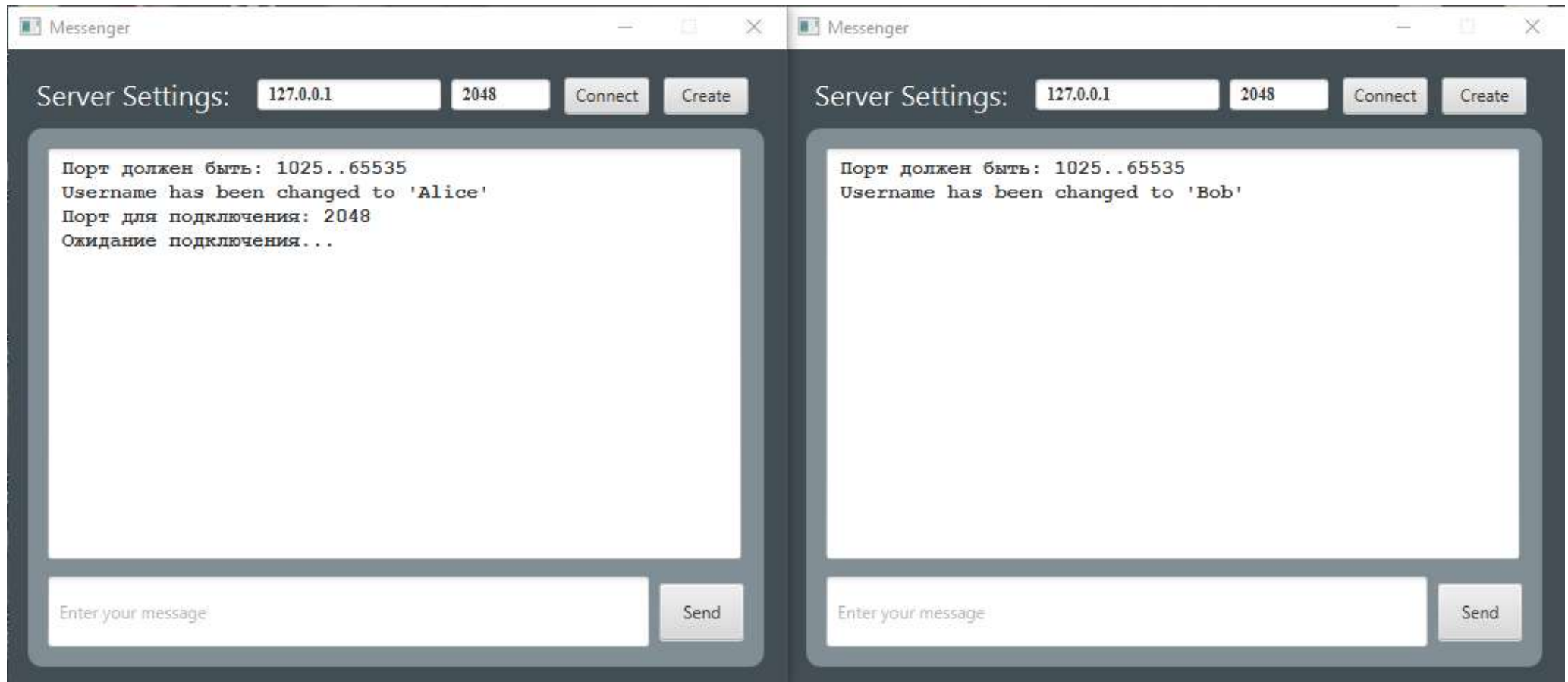
```

Шифрование и дешифровка через XOR, путем преобразования данных:

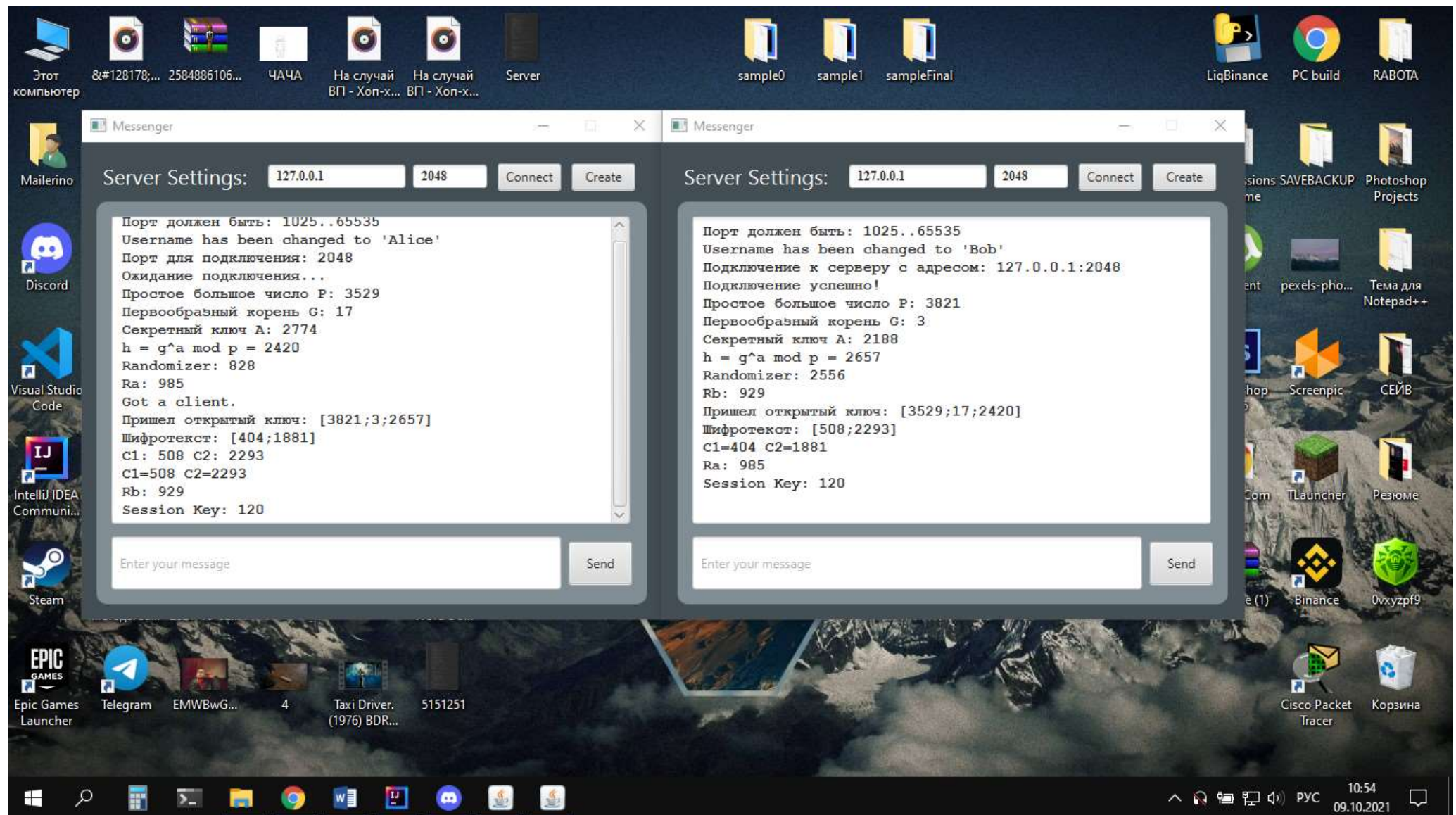
```
105 @ public static String Encryption (String Text, int Key){
106     String code = "";
107     char m;
108     for(int i = 0; i < Text.length(); i++){
109         char t = Text.charAt(i);
110         int n = t^Key;
111         m = (char)Integer.parseInt(String.valueOf(n));
112         code += m;
113     }
114     return code;
115 }
116 @ public static String Decryption (String Text, int Key){
117     String code = "";
118     char m;
119     for(int i = 0; i < Text.length(); i++){
120         char t = Text.charAt(i);
121         int n = t^Key;
122         m = (char)Integer.parseInt(String.valueOf(n));
123         code += m;
124     }
125     return code;
126 }
```


Примеры выполнения программы:

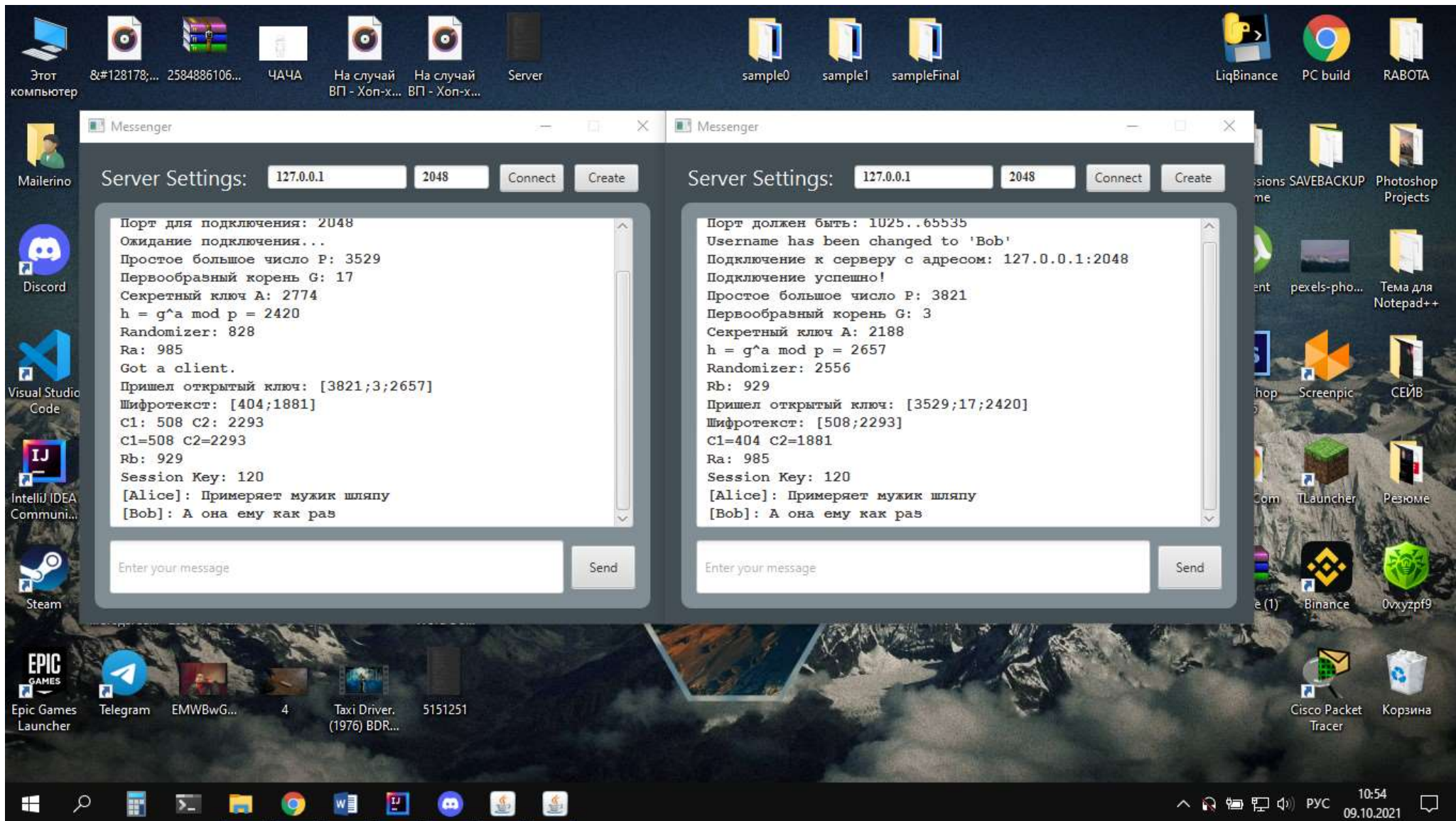
1. Поменяли ник и порт, потом запускаем Алису (сервер)

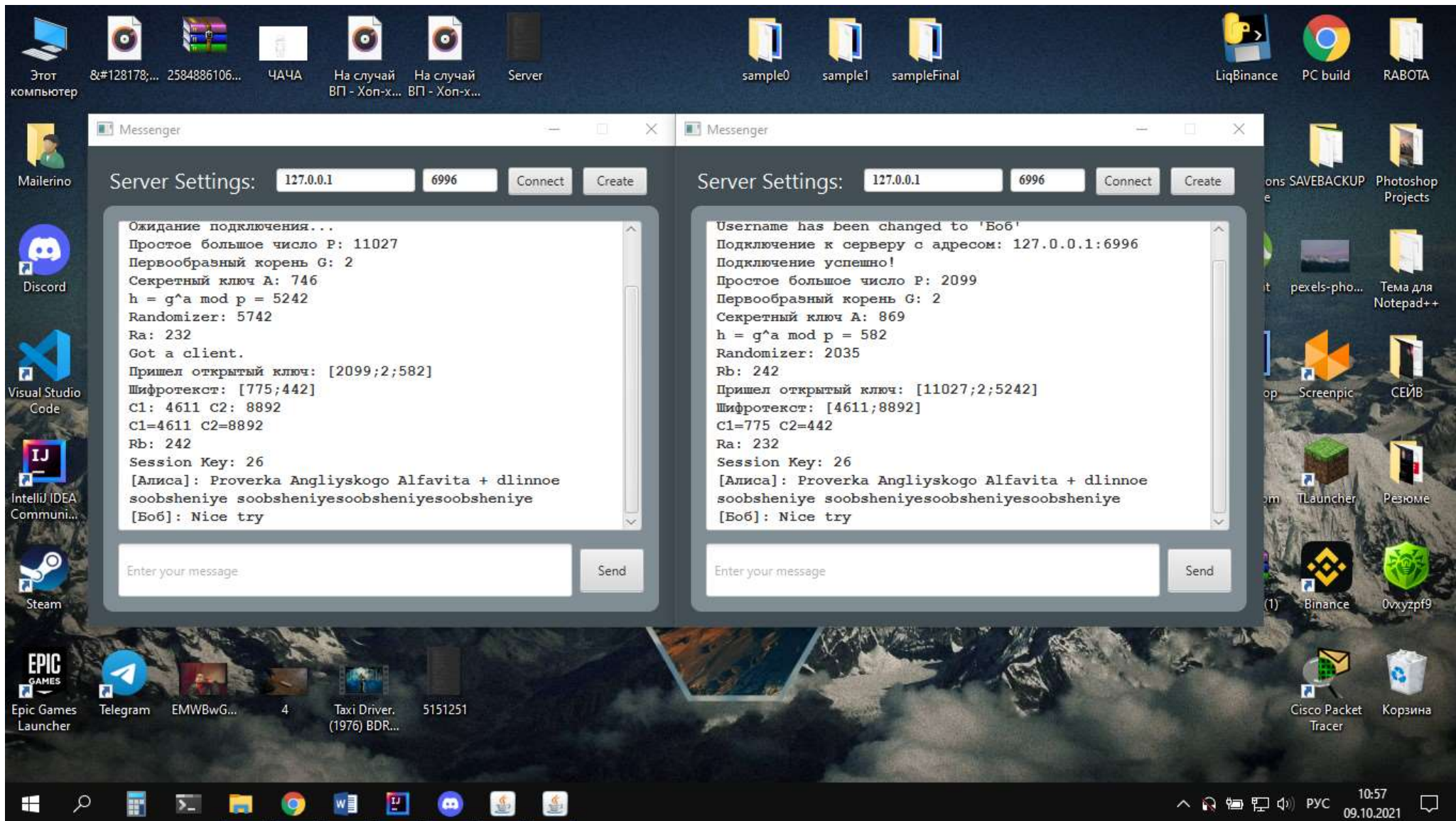


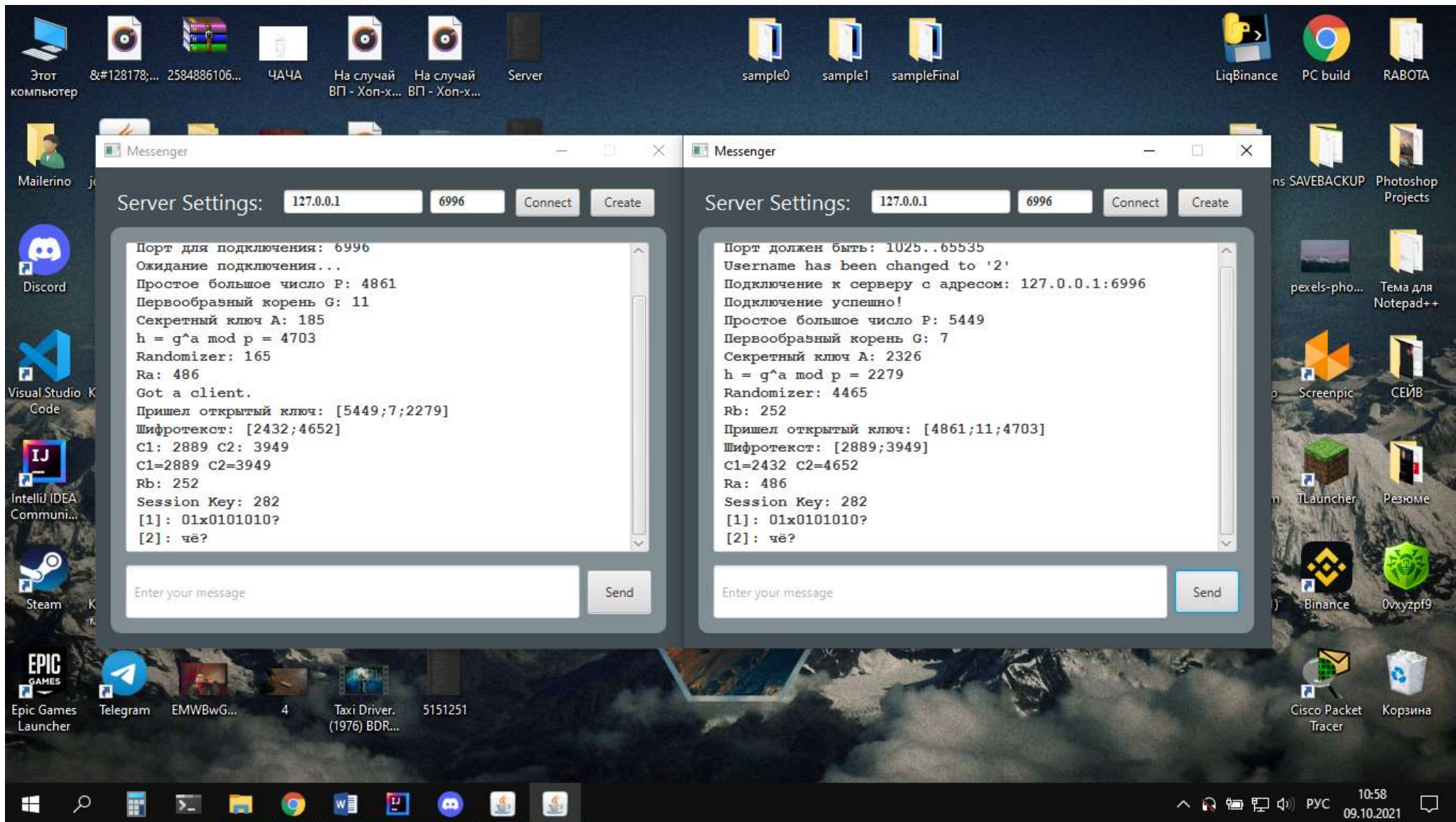
Создаем Боба (клиент) и подключаемся



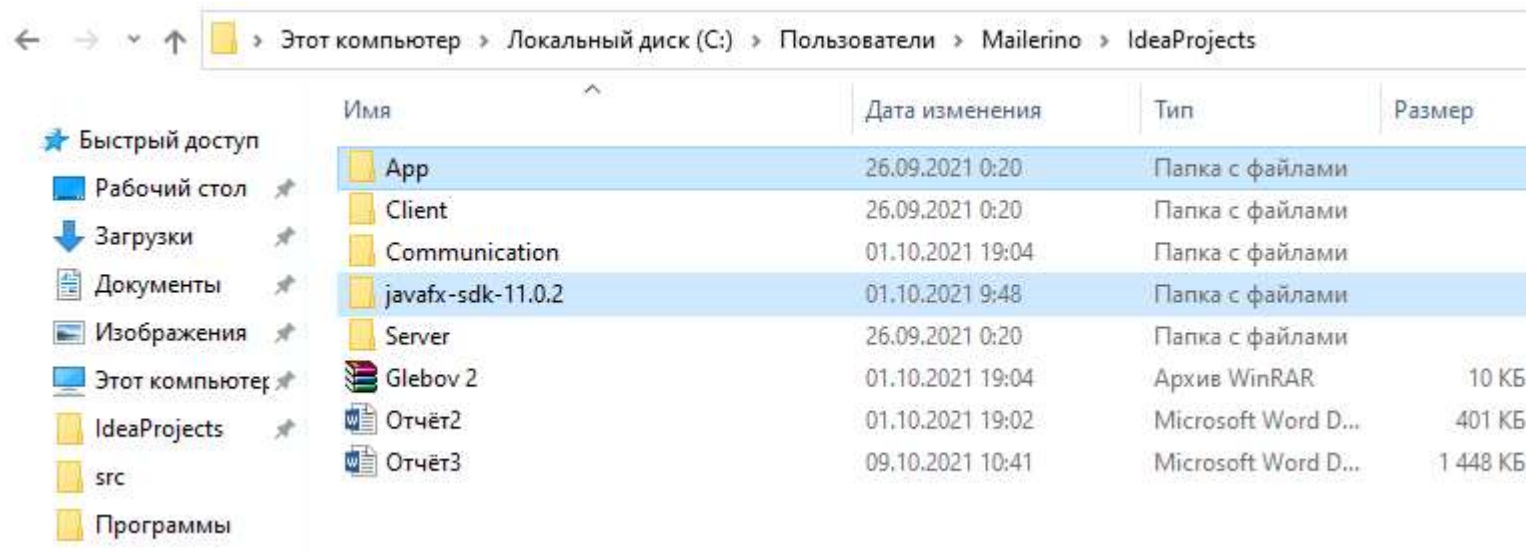
Прошла инициализация, сессионный ключ совпадает, можем обмениваться сообщениями







! Иногда при компиляции «ломается» программа, а точнее javafx через который она запускается:



Имя	Дата изменения	Тип	Размер
App	26.09.2021 0:20	Папка с файлами	
Client	26.09.2021 0:20	Папка с файлами	
Communication	01.10.2021 19:04	Папка с файлами	
javafx-sdk-11.0.2	01.10.2021 9:48	Папка с файлами	
Server	26.09.2021 0:20	Папка с файлами	
Glebov 2	01.10.2021 19:04	Архив WinRAR	10 КБ
Отчёт2	01.10.2021 19:02	Microsoft Word D...	401 КБ
Отчёт3	09.10.2021 10:41	Microsoft Word D...	1 448 КБ

Выражается в отсутствии скrolла при превышении высоты поля chat_field, в этот момент в intelliJ idea обилие ошибок по javafx.

То есть по сути обмен сообщениями происходит, но посмотреть полноценно чат ты не можешь так как нет скrolла.