

Лабораторная работа № 5

«Криптографические протоколы аутентификации, использующие криптографию с открытым ключом»

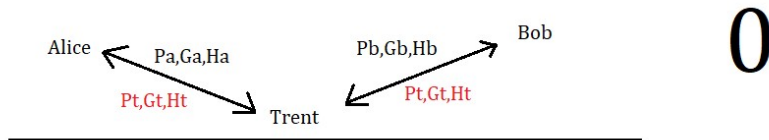
Отчет по лабораторной работе:

8. Напишите приложение с графическим интерфейсом, реализующее криптопротокол *Woo-Lam*. В качестве алгоритма шифрования с открытым ключом используйте схему Эль-Гамала.

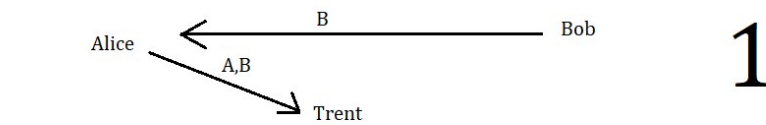
Алгоритм Шифрования – Эль Гамаль

Алгоритм Цифровой Подписи – Эль Гамаль

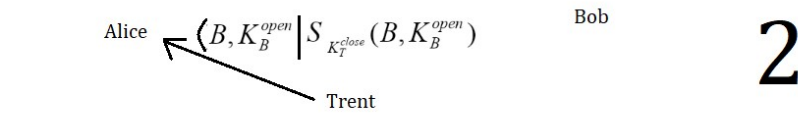
Woo-Lam



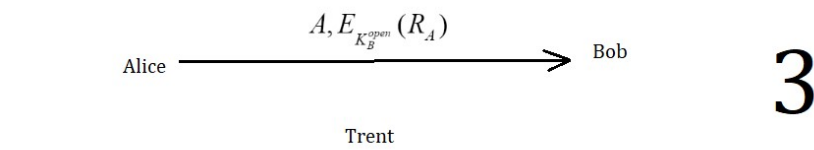
0



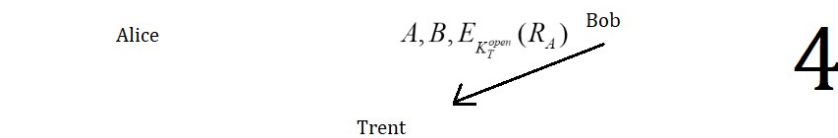
1



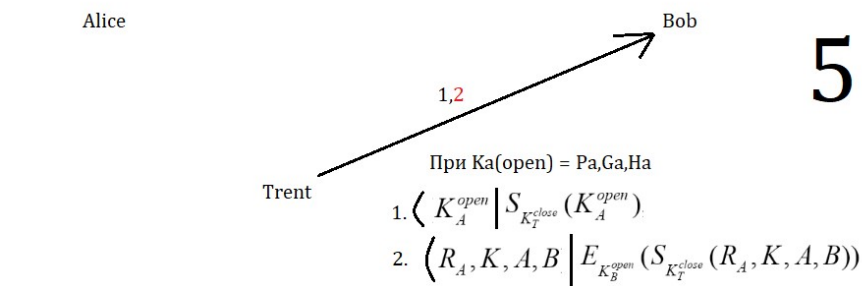
2



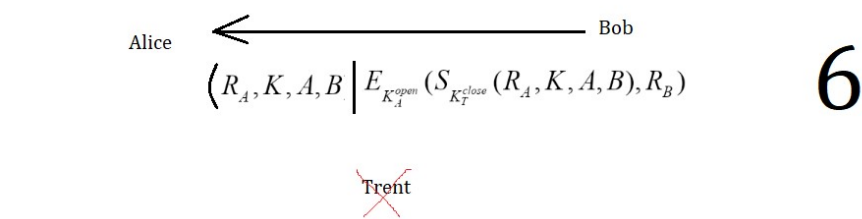
3



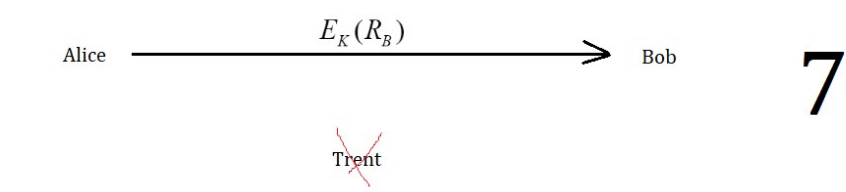
4



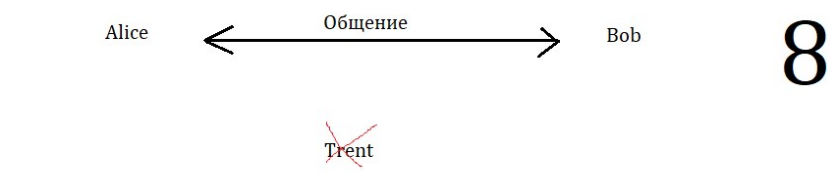
5



6



7



8

Криптопротокол 3.3.

Woo-Lam

Алиса и Боб генерируют пары «открытый ключ / закрытый ключ» и открытые ключи отправляют Тренту. Трент отправляет Алисе и Бобу свой открытый ключ.

1. Алиса посылает Тренту сообщение, состоящее из ее имени и имени Боба: A, B .

2. Трент посылает Алисе открытый ключ Боба K_B^{open} , подписанный закрытым ключом Трента K_T^{close} : $S_{K_T^{close}}(B, K_B^{open})$.

3. Алиса проверяет подпись Трента. Затем она посылает Бобу свое имя A и случайное число R_A , шифрованное открытым ключом Боба K_B^{open} : $A, E_{K_B^{open}}(R_A)$.

4. Боб посылает Тренту свое имя B , имя Алисы A и случайное число Алисы R_A , шифрованное открытым ключом Трента K_T^{open} : $A, B, E_{K_T^{open}}(R_A)$.

5. Трент посылает Бобу открытый ключ Алисы K_A^{open} , подписанный закрытым ключом Трента K_T^{close} . Он также посылает Бобу случайное число Алисы R_A , случайный сеансовый ключ K , имена Алисы и Боба A, B , подписав все это закрытым ключом Трента K_T^{close} и зашифровав открытым ключом Боба K_B^{open} : $S_{K_T^{close}}(K_A^{open}), E_{K_B^{open}}(S_{K_T^{close}}(R_A, K, A, B))$.

6. Боб проверяет подписи Трента. Затем он посылает Алисе вторую часть сообщения Трента, полученного на этапе 5, и новое случайное число R_B , зашифровав все открытым ключом Алисы K_A^{open} : $E_{K_A^{open}}(S_{K_T^{close}}(R_A, K, A, B), R_B)$.

7. Алиса проверяет подпись Трента и свое случайное число. Затем она посылает Бобу его случайное число R_B , шифрованное сеансовым ключом K : $E_K(R_B)$.

8. Боб расшифровывает свое случайное число и проверяет, что оно не изменилось.

9. Алиса и Боб шифруют свои сообщения, используя сеансовый ключ K .

Трент

```
254 //-----//
255 @ public static String Encryption (String Text, int Key){...}
266 public static int Encryption_int (int Text, int Key){...}
271 @ public static String Decryption (String Text, int Key){...}
282 public static int Decryption_int (int Text, int Key){...}
287 //-----//
288 public static int Gen_C1 (int external_G, int R, int external_P){...}
292 public static int Gen_C2 (int M, int external_H, int R, int external_P){...}
296 public static int Dec_C (int C1, int C2, int P, int A){...}
303 public static int GetPRoot(int p) {...}
310 public static boolean IsPRoot(long p, long a) {...}
325 public static int modInverse(int a, int m){...}
331 public static int gcd(int a, int b) {...}
350 private static boolean TestRabinMiller(int number, int mod){...}
364 public static int power(int x, int y, int p) {...}
387 public static boolean isPrime(int n, int k) {...}
398 public static int random(int from, int to) { return from + (int) (Math.random() * to); }
401 public static int generationLargeNumber(){...}
409 //-----//
410 @ public static int [] Signature(String msg, int primitiveRoot, int primeNumber, int closeKey ){...}
```

Функции:

Encryption/Encryption_int = шифрование string/int

Decryption/Decryption_int = дешифрование string/int

Gen_C1/Gen_C2 = генерация двух частей шифрованного сообщения по алгоритму Эль Гамала (Более подробно в отчёте 3 Лабораторной)

Dec_C = расшифровка сообщения по Эль Гамалю

GetPRoot + IsPRoot = поиск первообразного (примитивного) корня

modInverse = обратный в кольце по модулю

gdc = НОД двух чисел

TestRabinMiller = тест на простоту

Power = возведение в степень по модулю

isPrime = проверка на простоту (связана с тестом Рабина Миллера)

random = рандомизатор

generationLargeNumber = генерация большого простого числа (имеет внутри ограничение на max, его можно подвинуть и сделать криптосистему более устойчивой к взлому)

Signature = Функция подписи сообщения

```
1 package com.company;
2
3 import javax.annotation.processing.SupportedSourceVersion;
4 import java.net.ServerSocket;
5 import java.net.Socket;
6 import java.io.*;
7 import java.util.HashSet;
8 import java.util.Scanner;
9 import java.net.*;
10 import java.util.Set;
11
12 public class Main {
13     private static ServerSocket ss = null;
14     private static ServerSocket ss2 = null;
15     private static Socket socket = null;
16     private static Socket socket2 = null;
17     private static int Port = 8560; // Порт Боба
18     private static int Port2 = 8561; // Порт Алисы
19     private static int Session_Key = (int)(Math.random()*((3300-1300)+1))+1300; // Сеансовый ключ
20     public String internal_username = "Trent";
21     private static String aliceName = "";
22     private static String bobName = "";
23     private static int K = 0; // Переменная для цифровой подписи
24     private static int P = 0, G = 0, A = 0, H = 0, R = 0; // Компоненты Трента
25     private static int a_P = 0, a_G = 0, a_A = 0, a_H = 0, a_R = 0; // Компоненты Алисы
26     private static int b_P = 0, b_G = 0, b_A = 0, b_H = 0, b_R = 0; // Компоненты Боба
27     /* P - Простое Большое Число
28     * G - Примитивный корень по модулю P
29     * A - Секретный ключ
30     * R - Рандомайзер
31     * H - Вычисляемая функция */
32
33 }
```

```

34 ▶ public static void main(String[] args) {
35     try {
36         //Производим начальные вычисления
37         System.out.println("Session key: " + Session_Key);
38         //Генерация открытого ключа
39         P = generationLargeNumber();
40         System.out.println("Простое большое число P: " + P);
41         G = GetPRoot(P);
42         System.out.println("Первообразный корень G: " + G);
43         A = (int)(Math.random()*((P-2)-1)+1)+1; // (Math.random()*((max-min)+1))+min;
44         System.out.println("Секретный ключ A: " + A);
45         H = power(G,A,P); // Возведение в степень по модулю
46         System.out.println("h = g^a mod p = " + H);
47         // Генерируем рандомизатор r - целое число из [1;p-1]
48         R = (int)(Math.random()*((P-1)-1)+1)+1;
49         System.out.println("Randomizer: " + R);
50         for(int k=100; k < P-1; k++){
51             if((gcd(k,(P-1))) == 1){
52                 K = k; // Записываем взаимно простое число в K
53                 k = P-1; // Выходим из for
54             }
55         }
56         System.out.println("Случайное целое K: " + K);
57
58         ss = new ServerSocket(Port); // создаем сокет сервера и привязываем его к порту Боба
59         ss2 = new ServerSocket(Port2); // создаем сокет сервера и привязываем его к порту Алисы
60         System.out.println("Waiting Alice connection...");
61         socket2 = ss2.accept(); // заставляем сервер ждать подключений и выводим сообщение когда кто-то связался с сервером
62         System.out.println("Alice has been connected to Trent!");

```



```

63
64 // Конвертируем потоки в другой тип. Берем входной и выходной потоки сокета, теперь можем получать и отсылать данные клиенту.
65 InputStream sin2 = socket2.getInputStream();
66 OutputStream sout2 = socket2.getOutputStream();
67 DataInputStream in2 = new DataInputStream(sin2);
68 DataOutputStream out2 = new DataOutputStream(sout2);
69
70 a_P = in2.readInt(); // Получаем открытый ключ Алисы
71 a_G = in2.readInt();
72 a_H = in2.readInt();
73 System.out.println("Был получен открытый ключ Алисы: P=" + a_P + " G=" + a_G + " H=" + a_H);
74
75 out2.writeInt(P); // Отправляем 3 компоненты своего открытого ключа Алисе
76 out2.writeInt(G);
77 out2.writeInt(H);
78 System.out.println("\nОткрытый ключ был отправлен Алисе");
79
80 //Подключаем Боба
81 System.out.println("Waiting Bob connection...");
82 socket = ss.accept(); // заставляем сервер ждать подключений и выводим сообщение когда кто-то связался с сервером
83 System.out.println("Bob has been connected!");
84
85 // Берем входной и выходной потоки сокета, теперь можем получать и отсылать данные клиенту.
86 InputStream sin = socket.getInputStream();
87 OutputStream sout = socket.getOutputStream();
88 DataInputStream in = new DataInputStream(sin);
89 DataOutputStream out = new DataOutputStream(sout);
90
91 b_P = in.readInt(); // Получаем открытый ключ Боба
92 b_G = in.readInt();
93 b_H = in.readInt();
94 System.out.println("Был получен открытый ключ Боба: P=" + b_P + " G=" + b_G + " H=" + b_H);
95

```



```
96 out.writeInt(P); // Отправляем 3 компоненты своего открытого ключа Бобу
97 out.writeInt(G);
98 out.writeInt(H); // Отправляем H открытого ключа Трента Бобу для проверки подлинности сообщения
99 System.out.println("Открытый ключ был отправлен Бобу");
100
101 aliceName = in2.readUTF(); // Получаем сообщение от Алисы
102 bobName = in2.readUTF();
103 System.out.println("\nПришли имена: " + aliceName + " и " + bobName); // Шаг 1 закончен
104
105 System.out.println("bob_P: " + b_P + " hash is " + Integer.toString(b_P).hashCode());
106 System.out.println("bob_G: " + b_G + " hash is " + Integer.toString(b_G).hashCode());
107 System.out.println("bob_H: " + b_H + " hash is " + Integer.toString(b_H).hashCode());
108
109 out2.writeInt(b_P); // 1/3
110 out2.writeInt(b_G); // 2/3
111 out2.writeInt(b_H); // Отправили открытый ключ Боба 3/3 Алисе
112
113 int temp[] = new int[2];
114 temp = Signature(Integer.toString(b_P), G, P, A); // hashCode возвращает int, поэтому еще раз оборачиваем в string
115 System.out.println("Bob P after signature:");
116 for(int n = 0; n < temp.length; n++){System.out.println(" " + temp[n]);}
117 out2.writeInt(temp[0]);
118 out2.writeInt(temp[1]); // Отправляем 1/3 подписанный компонент открытого ключа Боба
119
120 temp = Signature(Integer.toString(b_G), G, P, A);
121 System.out.println("Bob G after signature:");
122 for(int n = 0; n < temp.length; n++){System.out.println(" " + temp[n]);}
123 out2.writeInt(temp[0]);
124 out2.writeInt(temp[1]); // Отправляем 2/3 подписанный компонент открытого ключа Боба
125
```

```

126     temp = Signature(Integer.toString(b_H), G, P, A);
127     System.out.println("Bob H after signature:");
128     for(int n = 0; n < temp.length; n++){System.out.println(" " + temp[n]);}
129     out2.writeInt(temp[0]);
130     out2.writeInt(temp[1]); // Отправляем 3/3 подписанный компонент открытого ключа Боба
131
132     // Шаг 4 получаем от Боба
133     String ex_B_name = in.readUTF(); // Имя Боба
134     String ex_A_name = in.readUTF(); // Имя Алисы
135     int ex_C1 = in.readInt();
136     int ex_C2 = in.readInt(); // Получили сообщение от Боба
137     if(aliceName.equals(ex_A_name)){System.out.println("Имя Алисы совпало с 1 шагом: " + ex_A_name);}
138     if(bobName.equals(ex_B_name)){System.out.println("Имя Боба совпало с 1 шагом: " + ex_B_name);}
139     System.out.println("Был получен шифротекст: [" + ex_C1 + ";" + ex_C2 + "]");
140     int external_R = Dec_C(ex_C1, ex_C2, P, A);
141     System.out.println("Ra: " + external_R);
142
143     //-----//
144     // Шаг 5 [1 Message]
145     System.out.println("Alice_P: " + a_P + " hash is " + Integer.toString(a_P).hashCode());
146     System.out.println("Alice_G: " + a_G + " hash is " + Integer.toString(a_G).hashCode());
147     System.out.println("Alice_H: " + a_H + " hash is " + Integer.toString(a_H).hashCode());
148
149     out.writeInt(a_P); // 1/3
150     out.writeInt(a_G); // 2/3
151     out.writeInt(a_H); // Отправили открытый ключ Алисы 3/3 Бобу
152
153     temp = Signature(Integer.toString(a_P), G, P, A);
154     System.out.println("Alice P after signature:");
155     for(int n = 0; n < temp.length; n++){System.out.println(" " + temp[n]);}
156     out.writeInt(temp[0]);
157     out.writeInt(temp[1]); // Отправляем 1/3 подписанный и зашифрованный компонент открытого ключа Алисы Бобу

```

```

158
159     temp = Signature(Integer.toString(a_G), G, P, A);
160     System.out.println("Alice G after signature:");
161     for(int n = 0; n < temp.length; n++){System.out.println(" " + temp[n]);}
162     out.writeInt(temp[0]);
163     out.writeInt(temp[1]); // Отправляем 2/3 подписанный и зашифрованный компонент открытого ключа Алисы Бобу
164
165     temp = Signature(Integer.toString(a_H), G, P, A);
166     System.out.println("Alice H after signature:");
167     for(int n = 0; n < temp.length; n++){System.out.println(" " + temp[n]);}
168     out.writeInt(temp[0]);
169     out.writeInt(temp[1]); // Отправляем 3/3 подписанный и зашифрованный компонент открытого ключа Алисы Бобу
170     //-----//
171     // Шаг 5 [2 Message] //-----DEFAULT-----//
172     out.writeInt(external_R); // Ra
173     out.writeInt(Session_Key); // SK
174     out.writeInt(v: (aliceName.hashCode())%P); // Отправили Alice name hashCode mod P
175     out.writeInt(v: (bobName.hashCode())%P); // Отправили Bob name hashCode mod P
176     //-----Signature + Encode-----//
177     temp = Signature(Integer.toString(external_R), G, P, A);
178     System.out.println("Ra after signature: ");
179     for(int n = 0; n < temp.length; n++){System.out.println(temp[n]);}
180     int C1 = Gen_C1(b_G, R, b_P);
181     int C2 = Gen_C2(temp[0], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
182     System.out.println("Ra1 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
183     out.writeInt(C1);
184     out.writeInt(C2); // Отправили шифротекст Ra1
185     //C1 = Gen_C1(b_G, R, b_P);
186     C2 = Gen_C2(temp[1], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
187     System.out.println("Ra2 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
188     //out.writeInt(C1);
189     out.writeInt(C2); // Отправили шифротекст Ra2
190     System.out.println("Шифротекст был отправлен"); // Отправляем Ra подписанный и зашифрованный компонент Трента Бобу
191

```



```

191     temp = Signature(Integer.toString(Session_Key),G,P,A);
192     System.out.println("Session Key after signature: ");
193     for(int n = 0; n < temp.length; n++){System.out.println(temp[n]);}
194     //C1 = Gen_C1(b_G, R, b_P);
195     C2 = Gen_C2(temp[0], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
196     System.out.println("Ra1 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
197     //out.writeInt(C1);
198     out.writeInt(C2); // Отправили шифротекст SK1
199     //C1 = Gen_C1(b_G, R, b_P);
200     C2 = Gen_C2(temp[1], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
201     System.out.println("Ra2 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
202     //out.writeInt(C1);
203     out.writeInt(C2); // Отправили шифротекст SK2
204     System.out.println("Шифротекст был отправлен"); // Отправляем Session Key подписанный и зашифрованный компонент Трента Бобу
205
206
207     temp = Signature(Integer.toString((String.valueOf(aliceName).hashCode())%P),G,P,A);
208     System.out.println("Alice hash name after signature: ");
209     for(int n = 0; n < temp.length; n++){System.out.println(temp[n]);}
210     //C1 = Gen_C1(b_G, R, b_P);
211     C2 = Gen_C2(temp[0], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
212     System.out.println("Alice name 1 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
213     //out.writeInt(C1);
214     out.writeInt(C2); // Отправили шифротекст AN1
215     //C1 = Gen_C1(b_G, R, b_P);
216     C2 = Gen_C2(temp[1], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
217     System.out.println("Alice name 2 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
218     //out.writeInt(C1);
219     out.writeInt(C2); // Отправили шифротекст AN2
220     System.out.println("Шифротекст был отправлен"); // Отправляем Alice name подписанный и зашифрованный компонент Трента Бобу
221

```

```

221
222     temp = Signature(Integer.toString( (String.valueOf(bobName).hashCode())%P),G,P,A);
223     System.out.println("Bob hash name after signature: ");
224     for(int n = 0; n < temp.length; n++){System.out.println(temp[n]);}
225     //C1 = Gen_C1(b_G, R, b_P);
226     C2 = Gen_C2(temp[0], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
227     System.out.println("Bob name 1 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
228     //out.writeInt(C1);
229     out.writeInt(C2); // Отправили шифротекст BN1
230     //C1 = Gen_C1(b_G, R, b_P);
231     C2 = Gen_C2(temp[1], b_H, R, b_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
232     System.out.println("Bob name 2 Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
233     //out.writeInt(C1);
234     out.writeInt(C2); // Отправили шифротекст BN2
235     System.out.println("Шифротекст был отправлен"); // Отправляем Bob паве подписанный и зашифрованный компонент Трента Бобу
236
237
238
239
240
241     out.flush(); // заставляем поток закончить передачу данных.
242     out2.flush(); // заставляем поток закончить передачу данных.
243
244     String choice = "";
245     while(true) {
246         System.out.println("\nТрент завершил начальную инициализацию, закрыть программу? [Y/N]: ");
247         Scanner sc = new Scanner(System.in);
248         choice = sc.next();
249         if (choice.equals("Y")) { System.exit( status: 9); }
250         choice = "";
251     }
252 } catch(Exception x) { x.printStackTrace(); }
253 }

```

Алиса

```
public static int [] Signature(String msg, int primitiveRoot, int primeNumber, int closeKey ){
    // msg - Исходная строка primitiveRoot = G primeNumber = P closeKey = A
    int[] arraySignatureElGamal = new int[2];
    int hashMessage = Math.abs(msg.hashCode()); // Используем модуль т.к. хеш hashCode() может быть отрицательный
    int randomNumber = random(2, primeNumber - 1);
    int reverseRandomNumber = modInverse(randomNumber, m: primeNumber - 1);
    while(gcd(randomNumber, b: primeNumber - 1) != 1 || (randomNumber * reverseRandomNumber) % (primeNumber - 1) != 1)
    {
        randomNumber = random(2, primeNumber - 1);
        reverseRandomNumber = modInverse(randomNumber, m: primeNumber - 1);
    }
    arraySignatureElGamal[0] = power(primitiveRoot, randomNumber, primeNumber);
    int numberU = Math.floorMod((hashMessage - (closeKey * arraySignatureElGamal[0])), primeNumber - 1);
    arraySignatureElGamal[1] = (reverseRandomNumber * numberU) % (primeNumber - 1);
    return arraySignatureElGamal;
}

public static boolean isSignature(String msg, int calculationResult, int primitiveRoot, int primeNumber, int signatureR, int signatureS ){
    // msg - сообщение calculationResult - H primitiveRoot - G primeNumber - P signatureR = a signatureS = b
    if(signatureR < 0 || signatureR >= primeNumber || signatureS < 0 || signatureS >= (primeNumber - 1) ) return false;
    int hashMessage = Math.abs(msg.hashCode());
    int leftComposition = ( power(calculationResult, signatureR, primeNumber) *
        power(signatureR, signatureS, primeNumber)) % primeNumber;
    int rightComposition = power(primitiveRoot, hashMessage, primeNumber);
    return leftComposition == rightComposition;
}
```



```

290 private class ReadMsgServer extends Thread {
291     @Override
292     public void run() {
293         try {
294             socket = ss.accept(); // заставляем сервер ждать подключений
295             chat_field.appendText(s: "\nBob has been connected to Alice!");
296             InputStream sin = socket.getInputStream(); // Конвертируем потоки в другой тип, чтоб легче обрабатывать текстовые сообщения.
297             OutputStream sout = socket.getOutputStream();
298             DataInputStream in = new DataInputStream(sin);
299             DataOutputStream out = new DataOutputStream(sout);
300
301             //Генерация открытого ключа
302             P = generationLargeNumber();
303             chat_field.appendText(s: "\nПростое большое число P: " + P);
304             G = GetPRoot(P);
305             chat_field.appendText(s: "\nПервообразный корень G: " + G);
306             A = (int)(Math.random()*(((P-2)-1)+1))+1; // (Math.random()*((max-min)+1))+min;
307             chat_field.appendText(s: "\nСекретный ключ A: " + A);
308             H = power(G,A,P); // Возведение в степень по модулю
309             chat_field.appendText(s: "\nh = g^a mod p = " + H);
310             // Генерируем рандомизатор r - целое число из [1;p-1]
311             R = (int)(Math.random()*(((P-1)-1)+1))+1;
312             chat_field.appendText(s: "\nRandomizer: " + R);
313             chat_field.appendText(s: "\nRa: " + internal_R);
314
315             external_username = in.readUTF(); // Принимаем имя Боба
316             chat_field.appendText(s: "\nПришло имя: " + external_username);
317             chat_field.appendText(s: "\nПодключение к Тренту: ");
318

```

```
320 socket2 = new Socket( host: "127.0.0.1", port: 8561); // Подключаемся к тренту по специальному для Алисы и Трента порту
321 InputStream sin2 = socket2.getInputStream();
322 OutputStream sout2 = socket2.getOutputStream(); // Конвертируем потоки в другой тип, чтоб легче обрабатывать текстовые сообщения.
323 DataInputStream in2 = new DataInputStream(sin2);
324 DataOutputStream out2 = new DataOutputStream(sout2);
325 chat_field.appendText( s: "Успешно");
326
327 out2.writeInt(P); // Отправляем 3 компоненты своего открытого ключа тренту
328 out2.writeInt(G);
329 out2.writeInt(H);
330 chat_field.appendText( s: "\n Открытый ключ был отправлен Тренту");
331
332 t_P = in2.readInt(); // Получаем открытый ключ Трента
333 t_G = in2.readInt();
334 t_H = in2.readInt();
335 chat_field.appendText( s: "\n Был получен открытый ключ Трента: P=" + t_P + " G=" + t_G + " H=" + t_H);
336 // Получаем H открытого ключа Трента для проверки подлинности сообщения
337
338 out2.writeUTF(internal_username); // Отсылаем Тренту имя Алисы и Боба
339 out2.writeUTF(external_username);
340
341 ex_P = in2.readInt();
342 ex_G = in2.readInt();
343 ex_H = in2.readInt(); // Получили открытый ключ Боба от Трента
344 chat_field.appendText( s: "\nПришел открытый ключ Боба [P.G.H]: [" + ex_P + "." + ex_G + "." + ex_H + "]);
```

```
345
346     int t_ex_P1 = in2.readInt(); // 1 часть подписи
347     int t_ex_P2 = in2.readInt(); // 2 часть подписи
348     int t_ex_G1 = in2.readInt();
349     int t_ex_G2 = in2.readInt();
350     int t_ex_H1 = in2.readInt();
351     int t_ex_H2 = in2.readInt(); // Получили 3 компоненты открытого ключа в подписанном виде
352     chat_field.appendText( s: "\nПроверяем подлинность P: " + isSignature(Integer.toString(ex_P), t_H, t_G, t_P, t_ex_P1, t_ex_P2));
353     chat_field.appendText( s: "\nПроверяем подлинность G: " + isSignature(Integer.toString(ex_G), t_H, t_G, t_P, t_ex_G1, t_ex_G2));
354     chat_field.appendText( s: "\nПроверяем подлинность H: " + isSignature(Integer.toString(ex_H), t_H, t_G, t_P, t_ex_H1, t_ex_H2));
355     if((isSignature(Integer.toString(ex_H), t_H, t_G, t_P, t_ex_H1, t_ex_H2)) == false){System.out.println("Подпись Трента не совпала [status: 4]"); System.exit( status: 4);}
356
357
358     out.writeUTF(internal_username); // Алиса отправляет Бобу своё имя
359     C1 = Gen_C1(ex_G, R, ex_P);
360     C2 = Gen_C2(internal_R, ex_H, R, ex_P); // Шифрование M={C1,C2}=Ra открытым ключом Боба
361     System.out.println("Ra Шифротекст от Алисы для Боба: [" + C1 + ";" + C2 + "]");
362     out.writeInt(C1);
363     out.writeInt(C2); // Отправили шифротекст ЗАКОНЧИЛИ ШАГ 3
```

```

364
365 // War 7
366 //-----DEFAULT-----//
367 int myNewRa = in.readInt();
368 Session_Key = in.readInt();
369 int myNewNameCode = in.readInt();
370 int exNewNameCode = in.readInt();
371 ex_R = in.readInt(); // Rb
372 chat_field.appendText( s: "\nПришло [Ra.Session Key.Rb]: [" + myNewRa + "." + Session_Key + "." + ex_R + "]" );
373 //-----Encode + Signature-----//
374 int en_sign_1 = in.readInt();
375 int en_sign_Ra1 = in.readInt();
376 int en_sign_Ra2 = in.readInt();
377 int en_sign_SK1 = in.readInt();
378 int en_sign_SK2 = in.readInt();
379 int en_sign_AN1 = in.readInt();
380 int en_sign_AN2 = in.readInt();
381 int en_sign_BN1 = in.readInt();
382 int en_sign_BN2 = in.readInt();
383 int en_Rb = in.readInt();
384 //-----Decryption-----//
385 int sign_Ra1 = Dec_C(en_sign_1, en_sign_Ra1, P, A);
386 int sign_Ra2 = Dec_C(en_sign_1, en_sign_Ra2, P, A);
387 int sign_SK1 = Dec_C(en_sign_1, en_sign_SK1, P, A);
388 int sign_SK2 = Dec_C(en_sign_1, en_sign_SK2, P, A);
389 int sign_AN1 = Dec_C(en_sign_1, en_sign_AN1, P, A);
390 int sign_AN2 = Dec_C(en_sign_1, en_sign_AN2, P, A);
391 int sign_BN1 = Dec_C(en_sign_1, en_sign_BN1, P, A);
392 int sign_BN2 = Dec_C(en_sign_1, en_sign_BN2, P, A);
393 int newRb = Dec_C(en_sign_1, en_Rb, P, A);

```

En_sign_* = зашифрованное_подписанное_что-то*

```
394 //-----Check-----//
395 chat_field.appendText( s: "\nПроверяем подлинность Ra: " + isSignature(Integer.toString(myNewRa), t_H, t_G, t_P, sign_Ra1, sign_Ra2));
396 chat_field.appendText( s: "\nПроверяем подлинность Session Key: " + isSignature(Integer.toString(Session_Key), t_H, t_G, t_P, sign_SK1, sign_SK2));
397 chat_field.appendText( s: "\nПроверяем подлинность Alice Name: " + isSignature(Integer.toString(myNewNameCode), t_H, t_G, t_P, sign_AN1, sign_AN2));
398 chat_field.appendText( s: "\nПроверяем подлинность Bob Name: " + isSignature(Integer.toString(exNewNameCode), t_H, t_G, t_P, sign_BN1, sign_BN2));
399 if((isSignature(Integer.toString(Session_Key), t_H, t_G, t_P, sign_SK1, sign_SK2)) == false) {
400     System.out.println("Подпись Трента Session Key не совпала [status: 6]");
401     System.exit( status: 6);
402 }
403 if(internal_R == myNewRa){chat_field.appendText( s: "\nRa совпало!");}else{chat_field.appendText( s: "\nRa НЕ совпало!");}
404
405 //-----7 to 8-----//
406 out.writeInt(Encryption_int(ex_R, Session_Key));
407
408
409
410
411 String line = null; //Создаем пустую строку "буфер"
412 while(true) {
413     line = in.readUTF(); // ожидаем пока клиент пришлет строку текста.
414     line = Decryption(line, Session_Key);
415     chat_field.appendText( s: "\n[" + external_username + "]: " + line);
416 }
417 } catch (IOException e) {
418     System.out.println("Исключение ReadMsgServer (Алисы)");
419 }
420 }
421 }
```


Боб

```
422 private class ReadMsgClient extends Thread {
423     @Override
424     public void run() {
425         try {
426             InputStream sin = socket.getInputStream(); // Конвертируем потоки в другой тип.
427             DataInputStream in = new DataInputStream(sin);
428             OutputStream sout = socket.getOutputStream();
429             DataOutputStream out = new DataOutputStream(sout);
430
431
432             //Генерация открытого ключа
433             P = generationLargeNumber();
434             chat_field.appendText(s: "\nПростое большое число P: " + P);
435             G = GetPRoot(P);
436             chat_field.appendText(s: "\nПервообразный корень G: " + G);
437             A = (int)(Math.random()*(((P-2)-1)+1))+1; // (Math.random()*((max-min)+1))+min;
438             chat_field.appendText(s: "\nСекретный ключ A: " + A);
439             H = power(G,A,P); // Возведение в степень по модулю
440             chat_field.appendText(s: "\nh = g^a mod p = " + H);
441             // Генерируем рандомизатор r - целое число из [1;p-1]
442             R = (int)(Math.random()*(((P-1)-1)+1))+1;
443             chat_field.appendText(s: "\nRandomizer: " + R);
444             chat_field.appendText(s: "\nRb: " + internal_R);
445
446             out.writeUTF(internal_username); // Отправляем Алисе имя Боба
447             chat_field.appendText(s: "\nАлисе было отправлено имя: " + internal_username);
448
```


Не обязательно ждать

```
449 Thread.sleep( millis: 3000);
450 chat_field.appendText( s: "\nПодключение к Тренту. ",
451 socket2 = new Socket( host: "127.0.0.1", port: 8560); // Подключаемся к специальному порту Трента (предназначенному Бобу)
452 OutputStream sout2 = socket2.getOutputStream();
453 InputStream sin2 = socket2.getInputStream();
454 DataOutputStream out2 = new DataOutputStream(sout2);
455 DataInputStream in2 = new DataInputStream(sin2);
456 chat_field.appendText( s: "Успешно");
457
458 out2.writeInt(P); // Отправляем 3 компоненты своего открытого ключа тренту
459 out2.writeInt(G);
460 out2.writeInt(H);
461 chat_field.appendText( s: "\n Открытый ключ был отправлен Тренту");
462
463 t_P = in2.readInt(); // Получаем открытый ключ Трента
464 t_G = in2.readInt();
465 t_H = in2.readInt();
466 chat_field.appendText( s: "\n Был получен открытый ключ Трента: P=" + t_P + " G=" + t_G + " H=" + t_H);
467
468
469 external_username = in.readUTF();
470 external_C1 = in.readInt();
471 external_C2 = in.readInt();
472 System.out.println("Ra шифротекст от Алисы: [" + external_C1 + ";" + external_C2 + "]");
473 int external_R = Dec_C(external_C1,external_C2,P,A);
474 chat_field.appendText( s: "\nRa Алисы: " + external_R);
475
476 // Шаг 4 Отправляем Тренту
477 out2.writeUTF(internal_username);
478 out2.writeUTF(external_username);
479 C1 = Gen_C1(t_G, R, t_P);
480 C2 = Gen_C2(external_R, t_H, R, t_P); // Шифрование M={C1,C2}=Ra открытым ключом Трента
481 System.out.println("Ra Шифротекст от Боба для Трента: [" + C1 + ";" + C2 + "]");
```

```

482 out2.writeInt(C1);
483 out2.writeInt(C2); // Отправили шифротекст
484 System.out.println("Шифротекст был отправлен");
485
486 //-----//
487 // Шаг 5 принимаем от Трента 1 сообщение
488 ex_P = in2.readInt();
489 ex_G = in2.readInt();
490 ex_H = in2.readInt(); // Открытый ключ Алисы в обычном виде
491 chat_field.appendText( s: "\nПришло [P.G.H]: [" + ex_P + "." + ex_G + "." + ex_H + "]");
492 int sign_P1 = in2.readInt(); // Получаем компоненты подписей
493 int sign_P2 = in2.readInt(); //
494 int sign_G1 = in2.readInt(); //
495 int sign_G2 = in2.readInt(); //
496 int sign_H1 = in2.readInt(); //
497 int sign_H2 = in2.readInt(); // Проверяем подписи
498 chat_field.appendText( s: "\nПроверяем подлинность P: " + isSignature(Integer.toString(ex_P), t_H, t_G, t_P, sign_P1, sign_P2));
499 chat_field.appendText( s: "\nПроверяем подлинность G: " + isSignature(Integer.toString(ex_G), t_H, t_G, t_P, sign_G1, sign_G2));
500 chat_field.appendText( s: "\nПроверяем подлинность H: " + isSignature(Integer.toString(ex_H), t_H, t_G, t_P, sign_H1, sign_H2));
501 if((isSignature(Integer.toString(ex_H), t_H, t_G, t_P, sign_H1, sign_H2)) == false){
502     System.out.println("Подпись Трента H не совпала [status: 4]");
503     System.exit( status: 4);
504 }
505 //-----//

```

```

506 //-----DEFAULT-----// 2 Сообщение
507 int newRa = in2.readInt();
508 Session_Key = in2.readInt();
509 int newAliceNameCode = in2.readInt();
510 int newBobNameCode = in2.readInt();
511 chat_field.appendText( s: "\nПришло [Ra.Session Key]: [" +newRa + "."+Session_Key+"]");
512 //-----Encode + Signature-----//
513 int en_sign_1 = in2.readInt();
514 int en_sign_Ra1 = in2.readInt();
515 int en_sign_Ra2 = in2.readInt();
516 int en_sign_SK1 = in2.readInt();
517 int en_sign_SK2 = in2.readInt();
518 int en_sign_AN1 = in2.readInt();
519 int en_sign_AN2 = in2.readInt();
520 int en_sign_BN1 = in2.readInt();
521 int en_sign_BN2 = in2.readInt();
522 //-----Decryption-----//
523 int sign_Ra1 = Dec_C(en_sign_1,en_sign_Ra1,P,A);
524 int sign_Ra2 = Dec_C(en_sign_1,en_sign_Ra2,P,A);
525 int sign_SK1 = Dec_C(en_sign_1,en_sign_SK1,P,A);
526 int sign_SK2 = Dec_C(en_sign_1,en_sign_SK2,P,A);
527 int sign_AN1 = Dec_C(en_sign_1,en_sign_AN1,P,A);
528 int sign_AN2 = Dec_C(en_sign_1,en_sign_AN2,P,A);
529 int sign_BN1 = Dec_C(en_sign_1,en_sign_BN1,P,A);
530 int sign_BN2 = Dec_C(en_sign_1,en_sign_BN2,P,A);
531 //-----Check-----//
532 chat_field.appendText( s: "\nПроверяем подлинность Ra: " + isSignature(Integer.toString(newRa),t_H,t_G,t_P,sign_Ra1,sign_Ra2));
533 chat_field.appendText( s: "\nПроверяем подлинность Session Key: " + isSignature(Integer.toString(Session_Key),t_H,t_G,t_P,sign_SK1,sign_SK2));
534 chat_field.appendText( s: "\nПроверяем подлинность Alice Name: " + isSignature(Integer.toString(newAliceNameCode),t_H,t_G,t_P,sign_AN1,sign_AN2));
535 chat_field.appendText( s: "\nПроверяем подлинность Bob Name: " + isSignature(Integer.toString(newBobNameCode),t_H,t_G,t_P,sign_BN1,sign_BN2));
536 if((isSignature(Integer.toString(newBobNameCode),t_H,t_G,t_P,sign_BN1,sign_BN2)) == false) {
537     System.out.println("Подпись Трента Bob Name не совпала [status: 5]");
538     System.exit( status: 5);
539 }

```



```

540 //-----6 to 7-----////-----DEFAULT-----//
541 out.writeInt(newRa);
542 out.writeInt(Session_Key);
543 out.writeInt(newAliceNameCode);
544 out.writeInt(newBobNameCode);
545 out.writeInt(internal_R); // Rb
546 //-----Encryption-----//
547 C1 = Gen_C1(ex_G, R, ex_P);
548 C2 = Gen_C2(sign_Ra1, ex_H, R, ex_P);
549 System.out.println("Ra1 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);|
550 out.writeInt(C1);
551 out.writeInt(C2); // Отправили шифротекст
552 C2 = Gen_C2(sign_Ra2, ex_H, R, ex_P);
553 System.out.println("Ra2 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);
554 out.writeInt(C2); // Отправили шифротекст
555 C2 = Gen_C2(sign_SK1, ex_H, R, ex_P);
556 System.out.println("SK1 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);
557 out.writeInt(C2); // Отправили шифротекст
558 C2 = Gen_C2(sign_SK2, ex_H, R, ex_P);
559 System.out.println("SK2 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);
560 out.writeInt(C2); // Отправили шифротекст
561 C2 = Gen_C2(sign_AN1, ex_H, R, ex_P);
562 System.out.println("AN1 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);
563 out.writeInt(C2); // Отправили шифротекст
564 C2 = Gen_C2(sign_AN2, ex_H, R, ex_P);
565 System.out.println("AN2 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);
566 out.writeInt(C2); // Отправили шифротекст
567 C2 = Gen_C2(sign_BN1, ex_H, R, ex_P);
568 System.out.println("BN1 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);
569 out.writeInt(C2); // Отправили шифротекст
570 C2 = Gen_C2(sign_BN2, ex_H, R, ex_P);
571 System.out.println("BN2 Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);
572 out.writeInt(C2); // Отправили шифротекст
573 C2 = Gen_C2(internal_R, ex_H, R, ex_P);
574 System.out.println("Rb Шифротекст от Боба для Алисы: [" + C1 + ";" + C2 + "]);

```

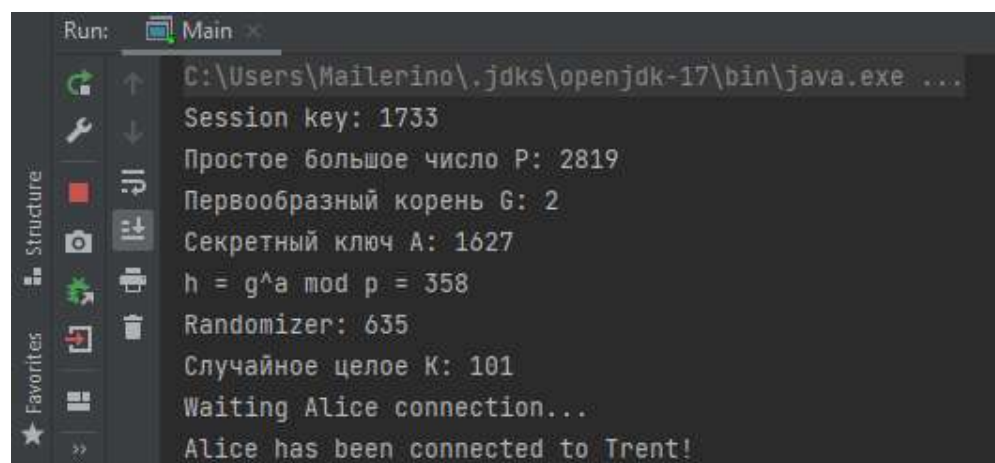
```

577 // Шаг 8
578 int en_myNewRb = in.readInt();
579 int myNewRb = Decryption_int(en_myNewRb, Session_Key);
580 if(myNewRb == internal_R){chat_field.appendText(s: "\n Rb совпало!");}else{chat_field.appendText(s: "\n Rb НЕ совпало!");}
581
582
583 String line = null; //Создаем пустую строку "буфер"
584 while(true) {
585     line = in.readUTF(); // ожидаем пока клиент пришлет строку текста.
586     line = Decryption(line, Session_Key);
587     chat_field.appendText(s: "\n[" + external_username + "]: " + line);
588 }
589 } catch (IOException | InterruptedException e) {
590     System.out.println("Исключение ReadMsgClient (Боб)");
591 }
592 }
593 }
594 }
595

```

Если подписи не совпадают, программа закрывается автоматически.

Тестирование программы:



```

Run: Main x
C:\Users\Mailerino\.jdk\openjdk-17\bin\java.exe ...
Session key: 1733
Простое большое число P: 2819
Первообразный корень G: 2
Секретный ключ A: 1627
h = g^a mod p = 358
Randomizer: 635
Случайное целое K: 101
Waiting Alice connection...
Alice has been connected to Trent!

```

Server Settings:

127.0.0.1

6996

Connect

Create

Порт должен быть: 1025..65535
Username has been changed to 'Алиса'
Порт для подключения: 6996
Ожидание подключения...
Bob has been connected to Alice!
Простое большое число P: 10369
Первообразный корень G: 13
Секретный ключ A: 9719
 $h = g^a \bmod p = 6649$
Randomizer: 4712
Ra: 460
Пришло имя: Боб
Подключение к Тренту: Успешно
Открытый ключ был отправлен Тренту
Был получен открытый ключ Трента: P=2819 G=2 H=358
Пришел открытый ключ Боба [P.G.H]: [3259.3.83]
Проверяем подлинность P: true

Enter your message

Send

Server Settings:

127.0.0.1

6996

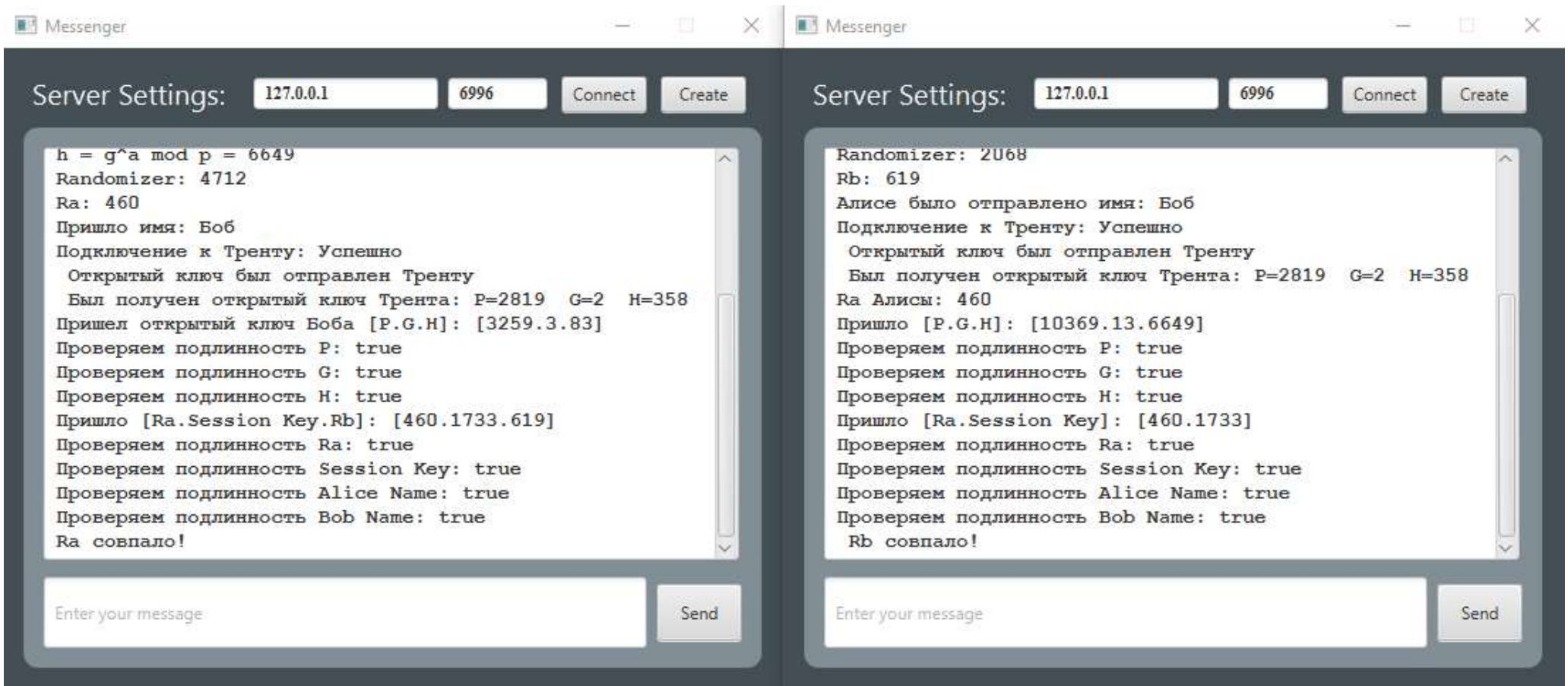
Connect

Create

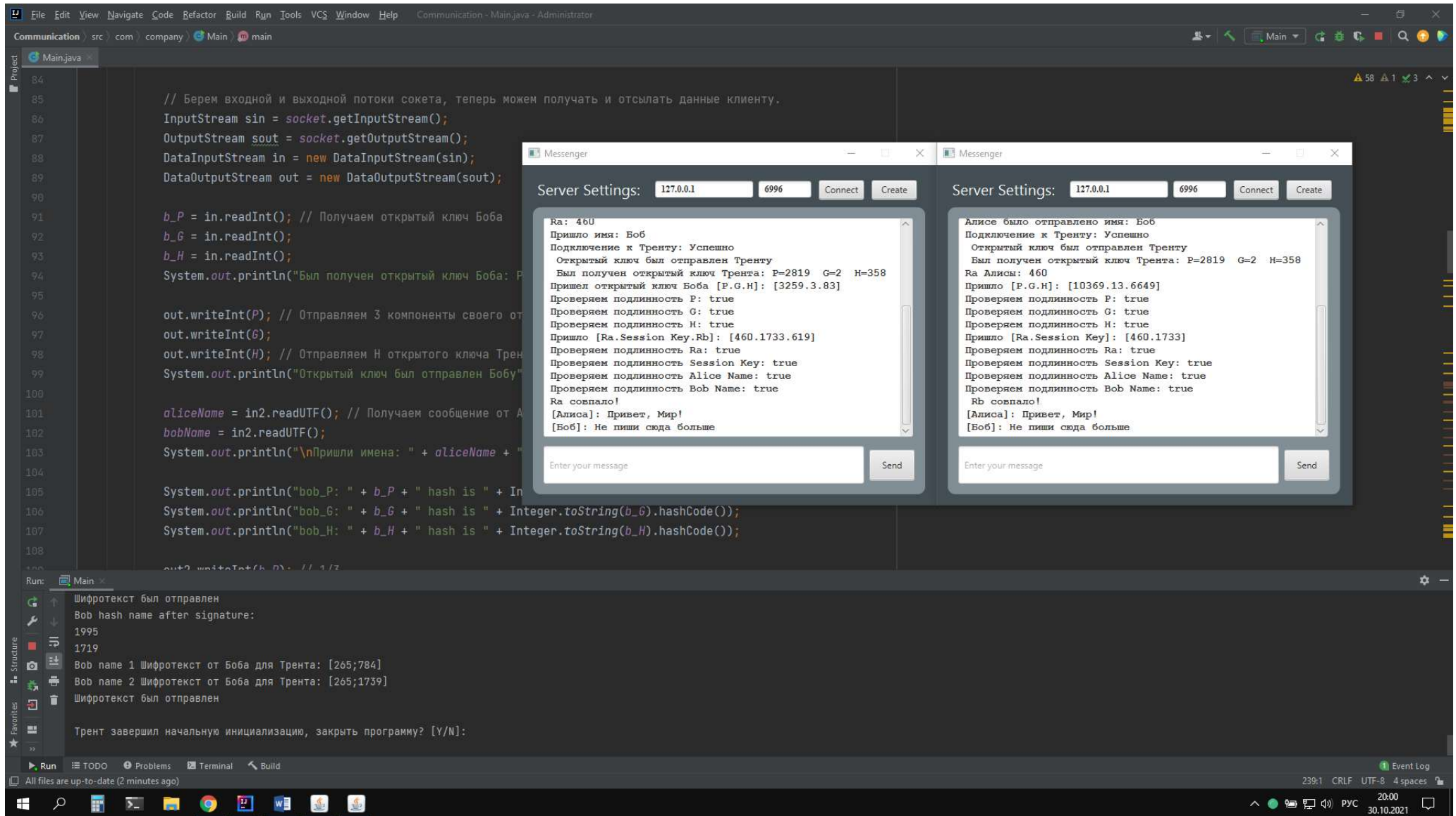
Порт должен быть: 1025..65535
Username has been changed to 'Боб'
Подключение к серверу с адресом: 127.0.0.1:6996
Подключение успешно!
Простое большое число P: 3259
Первообразный корень G: 3
Секретный ключ A: 2772
 $h = g^a \bmod p = 83$
Randomizer: 2068
Rb: 619
Алисе было отправлено имя: Боб
Подключение к Тренту: Успешно
Открытый ключ был отправлен Тренту
Был получен открытый ключ Трента: P=2819 G=2 H=358
Ra Алисы: 460
Пришло [P.G.H]: [10369.13.6649]
Проверяем подлинность P: true

Enter your message

Send



Подписи были подлинными, сессионные ключи от Трента совпадают, можно общаться:



Тестирование 2:

IDE interface showing a Java project named "Main.java" with code for a socket-based communication system. The code includes methods for reading and writing data over a socket, handling session keys, and verifying data integrity using hashes.

```
// Берем входной и выходной потоки сокета, теперь можем получать и отсылать данные клиенту.
InputStream sin = socket.getInputStream();
OutputStream sout = socket.getOutputStream();
DataInputStream in = new DataInputStream(sin);
DataOutputStream out = new DataOutputStream(sout);

b_P = in.readInt(); // Получаем открытый ключ Боба
b_G = in.readInt();
b_H = in.readInt();
System.out.println("Был получен открытый ключ Боба: P=" + b_P + " G=" + b_G + " H=" + b_H);

out.writeInt(P); // Отправляем 3 компоненты своего открытого ключа
out.writeInt(G);
out.writeInt(H); // Отправляем H открытого ключа Трента
System.out.println("Открытый ключ был отправлен Бобу");

aliceName = in2.readUTF(); // Получаем сообщение от Алисы
bobName = in2.readUTF();
System.out.println("\nПришли имена: " + aliceName + " и " + bobName);

System.out.println("bob_P: " + b_P + " hash is " + Integer.toString(b_P.hashCode()));
System.out.println("bob_G: " + b_G + " hash is " + Integer.toString(b_G.hashCode()));
System.out.println("bob_H: " + b_H + " hash is " + Integer.toString(b_H.hashCode()));
```

Two "Messenger" windows are open, displaying server settings (IP: 127.0.0.1, Port: 6996) and a log of the communication process. The logs show the exchange of session keys, verification of data integrity, and the receipt of messages from Alice and Bob.

Run console output:

```
Run: Main
C:\Users\Main\jdk\openjdk-17\bin\java.exe ...
Session key: 2756
Простое большое число P: 5897
Первообразный корень G: 3
Секретный ключ A: 3732
h = g^a mod p = 3200
Randomizer: 1139
Случайное целое K: 101
Waiting Alice connection...
Alice has been connected to Trent!
```

Реализация:

<https://protect.htmlweb.ru/ecp.htm>

Пример. Выберем: числа $P = 11$, $G = 2$ и секретный ключ $X = 8$. Вычисляем значение открытого ключа:

$$Y = G^X \bmod P = 2^8 \bmod 11 = 3 \text{ ,}$$

Предположим, что исходное сообщение M характеризуется хэш-значением $m = 5$.

Для того чтобы вычислить цифровую подпись для сообщения M , имеющего хэш-значение $m = 5$, сначала выберем случайное целое число $K = 9$. Убедимся, что числа K и $(P-1)$ являются взаимно простыми. Действительно, $\text{НОД}(9, 10) = 1$. Далее вычисляем элементы a и b подписи:

$$a = G^K \bmod P = 2^9 \bmod 11 = 6 \text{ ,}$$

элемент b определяем, используя расширенный алгоритм Евклида:

$$m = X * a + K * b \pmod{P-1} \text{ ,}$$

При $m = 5$, $a = 6$, $X = 8$, $K = 9$, $P = 11$ получаем

$$5 = 8 * 6 + 9 * b \pmod{10}$$

или

$$9 * b = -43 \pmod{10} \text{ .}$$

Решение: $b = 3$. Цифровая подпись представляет собой пару: $a = 6$, $b = 3$. Далее отправитель передает подписанное сообщение. Приняв подписанное сообщение и открытый ключ $Y = 3$, получатель вычисляет хэш-значение для сообщения M : $m = 5$, а затем вычисляет два числа:

$$\begin{aligned} Y^a a^b \pmod{P} &= 3^6 * 6^3 \pmod{11} = 10 \pmod{11}; \\ G^m \pmod{P} &= 2^5 \pmod{11} = 10 \pmod{11} \text{ .} \end{aligned}$$

Так как эти два целых числа равны, принятое получателем сообщение признается подлинным.