МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»

ЛАБОРАТОРНАЯ РАБОТА №7

на тему: *«Создание и использование цифровых сертификатов»*

Направление подготовки 09.03.03 «Прикладная информатика»
Профиль «Корпоративные информационные системы»
Дисциплина «Защита информации»

Выполнил:

студентка группы 201-361

Саблина Анна Викторовна

Проверил:

Харченко Елена Алексеевна

Москва 2023

Теоретическая часть

Сертификат – это цифровой документ, который используется для подтверждения подлинности и идентификации субъекта в сети. Он содержит информацию о субъекте (например, имя, адрес электронной почты, публичный ключ) и электронную подпись удостоверяющего центра (Центра сертификации), которая гарантирует подлинность содержащихся данных.

Сертификаты широко применяются в криптографии и сетевой безопасности. Они позволяют проверять, что публичный ключ, принадлежащий определенному субъекту, действительно принадлежит этому субъекту, и что информация, передаваемая через сеть, не была изменена в процессе передачи.

Удостоверяющий центр (Центр сертификации) выступает в роли доверенного третьего лица, которое выпускает и подписывает сертификаты. Он связывает публичный ключ с определенным субъектом, подтверждая подлинность этого связывания с помощью электронной подписи.

При установлении безопасного соединения между клиентом и сервером в сети, клиент обычно запрашивает сертификат у сервера, чтобы проверить его подлинность и убедиться, что соединение безопасно и надежно.

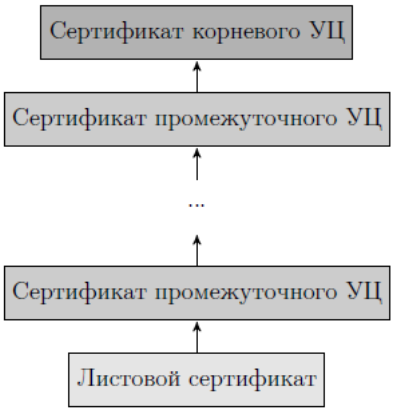

Рисунок 1 – Цепочка сертификатов

Вот пошаговый алгоритм для реализации простого клиент-серверного приложения с удостоверяющим центром, где клиенты могут обмениваться подписанными документами, используя RSA шифрование:

Шаги на стороне сервера:

1. Создается открытый и закрытый ключи RSA для сервера.
2. Публикуется открытый ключ сервера, чтобы клиенты могли получить к нему доступ.

Шаги на стороне клиента:

1. Создается открытый и закрытый ключи RSA для клиента.
2. Публикуется открытый ключ клиента, чтобы сервер и другие клиенты могли получить к нему доступ.

Процесс обмена подписанными документами между клиентами:

1. Клиент-отправитель выбирает документ, который хочет отправить.
2. Клиент-отправитель использует свой закрытый ключ для создания цифровой подписи документа.
3. Клиент-отправитель отправляет документ и цифровую подпись клиенту-получателю.

Проверка подписи документа клиентом-получателем:

1. Клиент-получатель получает документ и цифровую подпись от клиента-отправителя.
2. Клиент-получатель извлекает открытый ключ клиента-отправителя из удостоверяющего центра (сервера).
3. Клиент-получатель использует открытый ключ клиента-отправителя для проверки цифровой подписи документа.
4. Если проверка проходит успешно, клиент-получатель знает, что документ не был изменен и подписан клиентом-отправителем.

Данный алгоритм предоставляет основы для создания простого клиент-серверного приложения с использованием RSA шифрования.

Практическая часть

Для программы, реализующей простое клиент-серверное приложение, в котором сервер выступает в качестве удостоверяющего центра (УЦ), а клиенты могут обмениваться подписанными документами с возможностью проверки подписей, была использована утилита `OpenSSL`.

Были созданы на языке Java:

- библиотека `dp.scsa`;
- проект, реализующий поведение клиента А (Алисы);
- проект, реализующий поведение клиента Б (Боба);
- проект, реализующий поведение сервера.

Общий принцип работы программы:

1. Создается и запускается сервер, генерируя полный набор файлов по корневому и промежуточному сертификатам, который ожидает подключения клиентов.
2. Подключается Боб, который генерирует пару ключей и делает запрос на подпись листового сертификата. Сервер подписывает сертификат по запросу и возвращает Бобу полный комплект сертификатов для верификации цепочки сертификатов. Боб верифицирует полученную цепочку.
3. Подключается Алиса, которая генерирует пару ключей и делает запрос на подпись листового сертификата. Сервер подписывает сертификат по запросу и возвращает Алисе полный комплект сертификатов для верификации цепочки сертификатов. Алиса верифицирует полученную цепочку.
4. Сервер закрывает сокет после обработки каждого клиента.

5. Боб создает новое соединение, выступая в качестве сервера при соединении Peer-to-Peer. Ожидает подключения Алисы.
6. Алиса подключается к Бобу и передает ему
    a) документ,
    b) цифровую подпись файла,
    c) листовой сертификат.
7. Боб их получает и верифицирует цифровую подпись.
8. Боб отправляет Алисе такой же набор файлов, а Алиса в свою очередь получает их и верифицирует цифровую подпись.
9. Так происходит в цикле, пока один из собеседников не откажется от выбора документа для отправки, тем самым закрыв сокет.

```java
import dp.scsa.Server;

import java.io.IOException;

public class MyServer {
    public static void main(String[] args) throws IOException {
        Server server = new Server();
        server.start(8888);
    }
}
```

Листинг 1 – MyServer.java: класс приложения

```java
import dp.scsa.Client;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

import static dp.scsa.Tools.chooseFile;

public class Alice {
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        Client client = new Client("Alice");
        Socket socket = client.connectToServer("localhost", 8888);
        client.createCertificate(socket);

        try {
            // Создаем сокет для подключения к Бобу
            Socket socketP2P = new Socket("localhost", 1234);

            // Получаем потоки ввода/вывода для обмена данными с Бобом
            ObjectOutputStream outputStream = new
ObjectOutputStream(socketP2P.getOutputStream());
            outputStream.writeObject(client.getClientLogin());
            ObjectInputStream inputStream = new
```

```
ObjectInputStream(socketP2P.getInputStream());
            String penFriend = (String) inputStream.readObject();

            // Цикл для отправки и получения файлов
            while (true) {
                String filePath = chooseFile();
                if (filePath == null) {
                    socketP2P.close();
                    break;
                }
                client.setMessage(filePath);
                client.createSignature();

                // Отправляем документ Бобу
                client.sendFiles(socketP2P);

                System.out.println();

                // Получаем документ от Боба
                if (!socketP2P.isClosed())
                    client.receiveFiles(socketP2P, penFriend);
                else
                    break;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Листинг 2 – Alice.java: класс приложения

```
import dp.scsa.Client;

import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class Bob {
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        Scanner scanner = new Scanner(System.in);

        Client client = new Client("Bob");
        client.setMessage("Dear Alice.pdf");
        Socket socket = client.connectToServer("localhost", 8888);
        client.createCertificate(socket);
        socket.close();

        try {
            // Создаем серверный сокет, чтобы ожидать подключения Алисы
            ServerSocket serverSocket = new ServerSocket(1234);
            // Ожидаем подключения Алисы
            Socket socketP2P = serverSocket.accept();

            // Получаем потоки ввода/вывода для обмена данными с Бобом
            ObjectOutputStream outputStream = new
ObjectOutputStream(socketP2P.getOutputStream());
            outputStream.writeObject(client.getClientLogin());
            ObjectInputStream inputStream = new
ObjectInputStream(socketP2P.getInputStream());
            String penFriend = (String) inputStream.readObject();

            // Цикл для отправки и получения файлов
```

```
            while (true) {
                System.out.println();

                // Получаем документ от Алисы
                if (!socketP2P.isClosed())
                    client.receiveFiles(socketP2P, penFriend);
                else
                    break;

                System.out.print("\nВведите путь к файлу для отправки: ");
                String filePath = scanner.nextLine();

                // Проверяем, был ли введен путь к файлу
                if (!filePath.isEmpty() && (new File(filePath)).exists()) {
                    System.out.println("Выбранный файл: " + filePath);
                } else {
                    System.out.println("Файл не выбран");
                    socketP2P.close();
                    break;
                }

                client.setMessage(filePath);
                client.createSignature();

                // Отправляем документ Алисе
                client.sendFiles(socketP2P);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Листинг 3 – Bob.java: класс приложения

```
package dp.scsa;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;

import static dp.scsa.Tools.*;

/**
 * Класс, представляющий серверную часть приложения.
 * Сервер принимает подключение клиентов и обрабатывает их запросы.
 * Также сервер генерирует ключевые пары и сертификаты для корневого и промежуточного
Удостоверяющих Центров (УЦ),
 * а также подписывает листовые сертификаты клиентов.
 */
public class Server {
    private static String rootCert;
    private static String intermediateCert;
    private final HashMap<String, Socket> connectedClients;

    /**
     * Конструктор класса Server.
     *
     * @throws IOException если возникают проблемы при генерации ключевой пары и
сертификатов
     */
    public Server() throws IOException {
        connectedClients = new HashMap<>();
        createFolder("--root");
```

```
        createFolder("--inter");
        createFolder("--leaf");
        generateRootKeyPair();
        generateRootCSR();
        releaseSignedRootCert();
    }

    /**
     * Генерирует пару ключей для корневого УЦ.
     *
     * @throws IOException если возникают проблемы при выполнении команды
     */
    private static void generateRootKeyPair() throws IOException {
        // Генерация ключевой пары корневого УЦ
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "genpkey", "-algorithm", "RSA", "-out", "--root/root_keypair.pem");
        executeCommand(builder);
    }

    /**
     * Создает запрос на сертификат (CSR) для корневого УЦ.
     *
     * @throws IOException если возникают проблемы при выполнении команды
     */
    private static void generateRootCSR() throws IOException {
        // Создание запроса на сертификат (CSR) для корневого УЦ
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "req", "-new", "-subj", "\"/CN=Root CA\"", "-addext",
                "\"basicConstraints=critical,CA:TRUE\"", "-key",
                "--root/root_keypair.pem", "-out", "--root/root_csr.pem");
        executeCommand(builder);
    }

    /**
     * Подписывает сертификат корневым УЦ.
     *
     * @throws IOException если возникают проблемы при выполнении команды
     */
    private static void releaseSignedRootCert() throws IOException {
        // Подписание сертификата корневым УЦ
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "x509", "-req", "-in", "--root/root_csr.pem", "-copy_extensions",
                "copyall", "-key", "--root/root_keypair.pem", "-days",
                "3650", "-out", "--root/root_cert.pem");
        executeCommand(builder);

        rootCert = convertPEMFileToString("--root/root_cert.pem");
    }

    /**
     * Генерирует пару ключей для промежуточного УЦ.
     *
     * @throws IOException если возникают проблемы при выполнении команды
     */
    private static void generateIntermediateKeyPair() throws IOException {
        // Генерация ключевой пары промежуточного УЦ
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "genpkey", "-algorithm", "RSA", "-out", "--
inter/intermediate_keypair.pem");
        executeCommand(builder);
    }

    /**
     * Создает запрос на сертификат (CSR) для промежуточного УЦ.
     *
     * @throws IOException если возникают проблемы при выполнении команды
     */
    private static void generateIntermediateCSR() throws IOException {
```

```
        // Создание запроса на сертификат (CSR) для промежуточного УЦ
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "req", "-new", "-subj", "\"/CN=Intermediate CA\"", "-addext",
                "\"basicConstraints=critical,CA:TRUE\"", "-key",
                "--inter/intermediate_keypair.pem", "-out", "--
inter/intermediate_csr.pem");
        executeCommand(builder);
    }

    /**
     * Подписывает промежуточный сертификат корневым УЦ.
     *
     * @throws IOException если возникают проблемы при выполнении команды
     */
    private static void releaseSignedIntermediateCert() throws IOException {
        // Подписание промежуточного сертификата корневым УЦ
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "x509", "-req", "-in", "--inter/intermediate_csr.pem", "-copy_extensions",
                "copyall", "-CA", "--root/root_cert.pem", "-CAkey", "--
root/root_keypair.pem",
                "-days", "3650", "-out", "--inter/intermediate_cert.pem");
        executeCommand(builder);

        intermediateCert = convertPEMFileToString("--inter/intermediate_cert.pem");
    }

    /**
     * Подписывает листовой сертификат промежуточным УЦ.
     *
     * @param leafCSR    CSR (Certificate Signing Request) листового сертификата
     * @param clientHash хеш клиента
     * @return подписанный листовой сертификат в виде строки
     * @throws IOException если возникают проблемы при выполнении команды
     */
    private static String releaseSignedLeafCert(String leafCSR, int clientHash) throws
IOException {
        String leafCSRFile = "--leaf/leaf_csr_" + clientHash + ".pem";
        convertStringToPEMFile(leafCSR, leafCSRFile);
        String leafCertFile = "--leaf/leaf_cert_" + clientHash + ".pem";

        // Подписание листового сертификата промежуточным УЦ
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "x509", "-req", "-in", leafCSRFile, "-copy_extensions",
                "copyall", "-CA", "--inter/intermediate_cert.pem", "-CAkey",
                "--inter/intermediate_keypair.pem", "-days", "3650",
                "-out", leafCertFile);
        executeCommand(builder);

        return convertPEMFileToString(leafCertFile);
    }

    /**
     * Запускает сервер на указанном порту.
     *
     * @param serverPort порт сервера
     */
    public void start(int serverPort) {
        try {
            ServerSocket serverSocket = new ServerSocket(serverPort);
            System.out.println("Сервер запущен. Ожидание подключения клиентов...");

            while (true) {
                Socket socket = serverSocket.accept();

                ObjectInputStream inputStream = new
ObjectInputStream(socket.getInputStream());
                String clientName = (String) inputStream.readObject();
                System.out.println("Подключено клиент: " + clientName + ", " +
```

```
socket.getInetAddress().getHostAddress());

                connectedClients.put(clientName, socket);

                // Создание и запуск нового потока для обработки клиента
                Thread clientThread = new Thread(new ClientHandler(socket, clientName));
                clientThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Обрабатывает клиента и выполняет необходимые операции с сертификатами.
     *
     * @param socket     объект Socket для обмена данными с клиентом
     * @param clientName имя клиента
     * @throws IOException            если возникают проблемы при обмене данными
     * @throws ClassNotFoundException если класс сертификатов не найден
     */
    private void handleClient(Socket socket, String clientName) throws IOException,
ClassNotFoundException {
        int clientHash = clientName.hashCode();

        generateIntermediateKeyPair();
        generateIntermediateCSR();
        releaseSignedIntermediateCert();

        String leafCSR = receiveLeafCSR(socket);
        String leafCert = releaseSignedLeafCert(leafCSR, clientHash);
        sendCertPack(socket, leafCert);
        socket.close();
    }

    /**
     * Получает сертификат (CSR) для листового сертификата от клиента.
     *
     * @param socket объект Socket для обмена данными с клиентом
     * @return сертификат (CSR) в виде строки
     * @throws IOException            если возникают проблемы при обмене данными
     * @throws ClassNotFoundException если класс сертификата не найден
     */
    private String receiveLeafCSR(Socket socket) throws IOException, ClassNotFoundException
{
        // Получение сертификата (CSR) для листового сертификата
        ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());
        return (String) inputStream.readObject();
    }

    /**
     * Отправляет набор сертификатов клиенту для верификации.
     *
     * @param socket   объект Socket для обмена данными с клиентом
     * @param leafCert сертификат листового узла
     * @throws IOException если возникают проблемы при обмене данными
     */
    private void sendCertPack(Socket socket, String leafCert) throws IOException {
        // Отправка набора сертификатов клиенту для верификации
        ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());
        String[] certs = new String[3];
        certs[0] = rootCert;
        certs[1] = intermediateCert;
        certs[2] = leafCert;
        outputStream.writeObject(certs);
    }
```

```
    /**
     * Внутренний класс для обработки клиентов в отдельных потоках.
     */
    private class ClientHandler implements Runnable {
        private final Socket socket;
        private final String clientName;

        public ClientHandler(Socket socket, String clientName) {
            this.socket = socket;
            this.clientName = clientName;
        }

        @Override
        public void run() {
            // Обработка подключенного клиента
            try {
                handleClient(socket, clientName);
                socket.close();
            } catch (IOException | ClassNotFoundException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

Листинг 4 – Server.java: класс библиотеки

```
package dp.scsa;

import java.io.*;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;

import static dp.scsa.Tools.*;

/**
 * Класс, представляющий клиентскую часть приложения.
 * Клиент подключается к серверу, обменивается сертификатами и сообщениями.
 * Генерирует ключевую пару, запрос на сертификат (CSR) и отправляет их серверу.
 * Также верифицирует полученные сертификаты.
 */
public class Client {
    private static String rootCert;
    private static String intermediateCert;
    private static String leafCert;
    private static String leafCSR;
    private static String clientLogin;
    private static String clientMessage;

    /**
     * Конструктор класса Client.
     *
     * @param name имя клиента
     * @throws IOException если возникают проблемы при генерации ключевой пары и запроса на
сертификат
     */
    public Client(String name) throws IOException {
        clientLogin = name;
        generateLeafKeyPair();
        generateLeafCSR();
    }

    /**
     * Отправляет запрос на сертификат (CSR) для листового сертификата на сервер.
     *
```

```
     * @param socket объект Socket для обмена данными с сервером
     * @throws IOException если возникают проблемы при отправке данных
     */
    private static void sendLeafCSR(Socket socket) throws IOException {
        ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());
        outputStream.writeObject(leafCSR);
    }

    /**
     * Получает набор сертификатов для верификации от сервера.
     *
     * @param socket объект Socket для обмена данными с сервером
     * @throws IOException            если возникают проблемы при получении данных
     * @throws ClassNotFoundException если класс сертификатов не найден
     */
    private static void receiveCertPack(Socket socket) throws IOException,
ClassNotFoundException {
        ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());
        String[] certs = (String[]) inputStream.readObject();
        rootCert = certs[0];
        intermediateCert = certs[1];
        leafCert = certs[2];
    }

    public String getClientLogin() {
        return clientLogin;
    }

    /**
     * Устанавливает сообщение и хэш-значение для клиентского сообщения.
     *
     * @param filePath путь к файлу сообщения
     */
    public void setMessage(String filePath) {
        clientMessage = filePath;
    }

    /**
     * Создает листовой сертификат и верифицирует его.
     *
     * @param socket сокет для обмена данными
     * @throws IOException            если возникают ошибки ввода-вывода при взаимодействии
с сокетом
     * @throws ClassNotFoundException если класс не найден при десериализации
     */
    public void createCertificate(Socket socket) throws IOException, ClassNotFoundException
{
        sendLeafCSR(socket);
        receiveCertPack(socket);
        verifyLeafCert();
        socket.close();
    }

    /**
     * Устанавливает соединение с сервером.
     *
     * @param serverAddress IP-адрес сервера
     * @param serverPort    порт сервера
     * @return объект Socket для обмена данными с сервером
     */
    public Socket connectToServer(String serverAddress, int serverPort) {
        try {
            Socket socket = new Socket(serverAddress, serverPort);
            System.out.println("Подключено к серверу: " +
socket.getInetAddress().getHostAddress());

            ObjectOutputStream outputStream = new
ObjectOutputStream(socket.getOutputStream());
```

```
            outputStream.writeObject(clientLogin);
            return socket;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * Отправляет файлы клиенту Б, включая сообщение,
     * цифровую подпись и листовой сертификат.
     *
     * @param socket объект Socket для обмена данными с клиентом Б
     */
    public void sendFiles(Socket socket) {
        try {
            // Подготовка файлов для отправки
            File[] filesToSend = {new File(clientMessage), new File("leaf_cert.pem"), new
File("signature.bin")};

            DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());
// Инициализируем outputStream

            // Отправка количества файлов
            outputStream.writeInt(filesToSend.length);
            outputStream.flush();

            // Отправка файлов
            for (File file : filesToSend) {
                sendFile(socket, file);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Отправляет файл клиенту Б.
     *
     * @param socket объект Socket для обмена данными с клиентом Б
     * @param file   отправляемый файл
     */
    private void sendFile(Socket socket, File file) throws IOException {
        DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());

        // Отправка имени файла и размера
        outputStream.writeUTF(file.getName());
        outputStream.writeLong(file.length());
        outputStream.flush();

        // Отправка содержимого файла
        FileInputStream fileInputStream = new FileInputStream(file);
        byte[] buffer = new byte[4096];
        int bytesRead;

        while ((bytesRead = fileInputStream.read(buffer)) != -1) {
            outputStream.write(buffer, 0, bytesRead);
        }

        outputStream.flush();
        System.out.println("Файл успешно отправлен: " + file.getName());

        fileInputStream.close();
    }

    /**
     * Получает файлы клиента Б, включая сообщение,
     * цифровую подпись и листовой сертификат.
```

```
     *
     * @param socket     объект Socket для обмена данными с клиентом Б
     * @param clientName имя клиента Б
     */
    public void receiveFiles(Socket socket, String clientName) {
        try {
            DataInputStream inputStream = new DataInputStream(socket.getInputStream());

            // Получение количества файлов от клиента
            int fileCount = inputStream.readInt();
            System.out.println("Количество файлов для получения: " + fileCount);

            // Создание папки для сохранения файлов, если она не существует
            String folderName = "received_files_" + clientName;
            createFolder(folderName);

            // Получение файлов от клиента
            String[] temp;
            String sign = "", cert = "", file = "";
            for (int i = 0; i < fileCount; i++) {
                temp = receiveFile(inputStream, folderName);
                switch (temp[0]) {
                    case ("s") -> sign = temp[1];
                    case ("c") -> cert = temp[1];
                    default -> file = temp[1];
                }
            }

            verifySignature(sign, cert, file);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Получает файл клиента Б.
     *
     * @param inputStream входной поток данных для чтения файла от клиента Б
     * @param folderName  имя папки, в которую будет сохранен файл
     * @throws IOException если возникают проблемы ввода-вывода при чтении или записи файла
     */
    private String[] receiveFile(DataInputStream inputStream, String folderName) throws
IOException {
        // Создание объекта для форматирования даты и времени
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss");

        // Получение информации о файле
        String fileRole = inputStream.readUTF();
        String fileName = dateFormat.format(new Date()) + "_" + fileRole;
        long fileSize = inputStream.readLong();
        System.out.println("Получение файла: " + fileName + " (" + fileSize + " байт)");

        // Создание файлового потока для записи файла
        String filePath = folderName + "/" + fileName;
        FileOutputStream fileOutputStream = new FileOutputStream(filePath);

        // Чтение и запись содержимого файла
        byte[] buffer = new byte[4096];
        int bytesRead;
        long totalBytesRead = 0;

        while (totalBytesRead < fileSize) {
            int bytesToRead = (int) Math.min(buffer.length, fileSize - totalBytesRead);
            bytesRead = inputStream.read(buffer, 0, bytesToRead);
            if (bytesRead == -1) {
                break;
            }
            fileOutputStream.write(buffer, 0, bytesRead);
```

```
            totalBytesRead += bytesRead;
        }

        System.out.println("Файл успешно получен: " + fileName);

        fileOutputStream.close();

        return switch (fileRole) {
            case ("signature.bin") -> new String[]{"s", "\"" + filePath + "\""};
            case ("leaf_cert.pem") -> new String[]{"c", "\"" + filePath + "\""};
            default -> new String[]{"f", "\"" + filePath + "\""};
        };
    }

    /**
     * Генерирует пару ключей для листового сертификата.
     *
     * @throws IOException если возникают проблемы при генерации пары ключей
     */
    private void generateLeafKeyPair() throws IOException {
        ProcessBuilder builder = new ProcessBuilder("openssl", "genpkey", "-algorithm",
"RSA", "-out", "leaf_keypair.pem");
        executeCommand(builder);
    }

    /**
     * Генерирует запрос на сертификат (CSR) для листового сертификата.
     *
     * @throws IOException если возникают проблемы при создании CSR
     */
    private void generateLeafCSR() throws IOException {
        ProcessBuilder builder = new ProcessBuilder("openssl", "req", "-new", "-subj",
"\"/CN=Leaf\"", "-addext", "\"basicConstraints=critical,CA:FALSE\"", "-key",
"leaf_keypair.pem", "-out", "leaf_csr.pem");
        executeCommand(builder);

        leafCSR = convertPEMFileToString("leaf_csr.pem");
    }

    /**
     * Проверяет листовой сертификат.
     *
     * @throws IOException если возникают проблемы при проверке сертификата
     */
    private void verifyLeafCert() throws IOException {
        convertStringToPEMFile(rootCert, "root_cert.pem");
        convertStringToPEMFile(intermediateCert, "intermediate_cert.pem");
        convertStringToPEMFile(leafCert, "leaf_cert.pem");

        // Проверка подписи листового сертификата
        // с использованием цепочки корневого и промежуточного сертификатов
        ProcessBuilder builder = new ProcessBuilder("openssl", "verify", "-verbose", "-
show_chain", "-trusted", "root_cert.pem", "-untrusted", "intermediate_cert.pem",
"leaf_cert.pem");
        executeReadableCommand(builder);
    }

    /**
     * Получает цифровую подпись документа.
     *
     * @throws IOException если возникают проблемы при получении цифровой подписи
     */
    public void createSignature() throws IOException {
        // Получение цифровой подписи
        ProcessBuilder builder = new ProcessBuilder("openssl", "dgst", "-sha512", "-sign",
"leaf_keypair.pem", "-out", "signature.bin", "\"" + clientMessage + "\"");
        executeCommand(builder);
    }
```

```
    /**
     * Проверяет цифровую подпись полученного документа.
     *
     * @throws IOException если возникают проблемы при проверке цифровой подписи
     */
    private void verifySignature(String sign, String cert, String file) throws IOException
{
        // Получение подписанного публичного ключа из листового сертификата
        ProcessBuilder builder = new ProcessBuilder("openssl",
                "x509", "-in", cert, "-pubkey", "-noout", "-out", "public_key.pem");
        executeCommand(builder);

        // Получение цифровой подписи
        builder = new ProcessBuilder("openssl",
                "dgst", "-sha512", "-verify", "public_key.pem", "-signature", sign, file);
        executeReadableCommand(builder);

        (new File("public_key.pem")).delete();
    }
}
```

Листинг 5 – Client.java: класс библиотеки

```
package dp.scsa;

import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;

/**
 * Класс, представляющий набор вспомогательных инструментов.
 */
public class Tools {
    /**
     * Выполняет openssl команду в системной оболочке.
     *
     * @param processBuilder объект ProcessBuilder для выполнения команды
     * @throws IOException если возникают проблемы при выполнении команды
     */
    public static void executeCommand(ProcessBuilder processBuilder) throws IOException {
        processBuilder.redirectErrorStream(true);
        Process process = processBuilder.start();
        try {
            process.waitFor();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    /**
     * Выполняет openssl команду в системной оболочке
     * и выводит содержимое консоли при выполнении.
     *
     * @param processBuilder объект ProcessBuilder для выполнения команды
     * @throws IOException если возникают проблемы при выполнении команды
     */
    public static void executeReadableCommand(ProcessBuilder processBuilder) throws
IOException {
        processBuilder.redirectErrorStream(true);
        Process process = processBuilder.start();

        // Создаем буфер для чтения вывода из консоли OpenSSL
```

```
        BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
        String line;

        // Читаем и выводим каждую строку из консоли OpenSSL в консоль IntelliJ IDEA
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }

        try {
            process.waitFor();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    /**
     * Преобразует содержимое файла PEM в строку.
     *
     * @param filePath путь к файлу PEM
     * @return содержимое файла PEM в виде строки
     */
    public static String convertPEMFileToString(String filePath) {
        try {
            List<String> lines = Files.readAllLines(Paths.get(filePath));
            return String.join("\n", lines);
        } catch (IOException e) {
            System.err.println("Ошибка при чтении файла PEM: " + e.getMessage());
        }
        return "";
    }

    /**
     * Записывает строку в файл PEM.
     *
     * @param pemString содержимое в формате PEM
     * @param filePath  путь к файлу PEM
     */
    public static void convertStringToPEMFile(String pemString, String filePath) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            writer.write(pemString);
        } catch (IOException e) {
            System.err.println("Ошибка при записи в файл PEM: " + e.getMessage());
        }
    }

    /**
     * Создает папку с указанным именем.
     *
     * @param folderName имя папки
     */
    public static void createFolder(String folderName) {
        File folder = new File(folderName);
        if (!folder.exists()) {
            folder.mkdir();
        }
    }

    /**
     * Выбирает файл с помощью диалогового окна JFileChooser.
     * Позволяет пользователю выбрать текстовые файлы с расширениями: txt, pdf, docx, rtf,
html, xml, json, csv.
     *
     * @return Путь к выбранному файлу в виде строки. Возвращает null, если файл не был
выбран.
     */
    public static String chooseFile() {
        System.out.println("\nВыберите файл для отправки");
```

```
        // Создаем экземпляр JFileChooser
        JFileChooser fileChooser = new JFileChooser();

        // Определяем фильтр файлов, если нужно
        FileNameExtensionFilter filter = new FileNameExtensionFilter("Текстовые файлы",
                "txt", "pdf", "docx", "rtf", "html", "xml", "json", "csv");
        fileChooser.setFileFilter(filter);

        // Задаем изначальную директорию
        String currentDirectory = System.getProperty("user.dir"); // Директория текущего
проекта
        fileChooser.setCurrentDirectory(new java.io.File(currentDirectory));

        // Открываем проводник для выбора файла
        int result = fileChooser.showOpenDialog(null);

        // Проверяем, был ли выбран файл
        if (result == JFileChooser.APPROVE_OPTION) {
            // Получаем выбранный файл
            java.io.File selectedFile = fileChooser.getSelectedFile();
            String filePath = selectedFile.getPath();
            System.out.println("Выбранный файл: " + filePath);

            return filePath;
        } else {
            System.out.println("Файл не выбран");
        }
        return null;
    }
}
```

Листинг 6 – Tools.java: класс библиотеки

Наглядный пример использования программы:

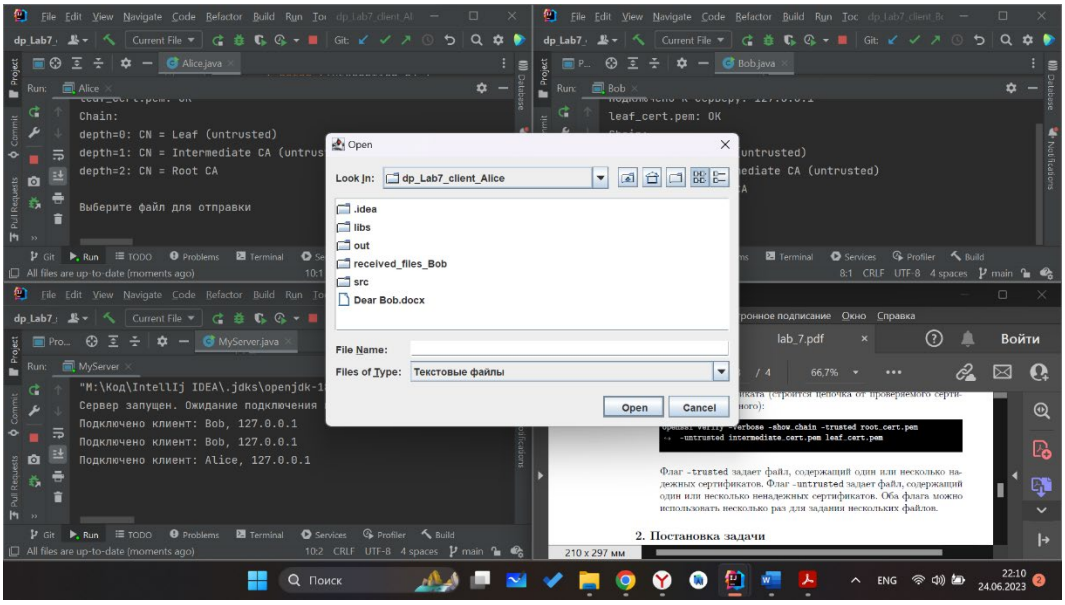

Рисунок 2 – Пример работы программы

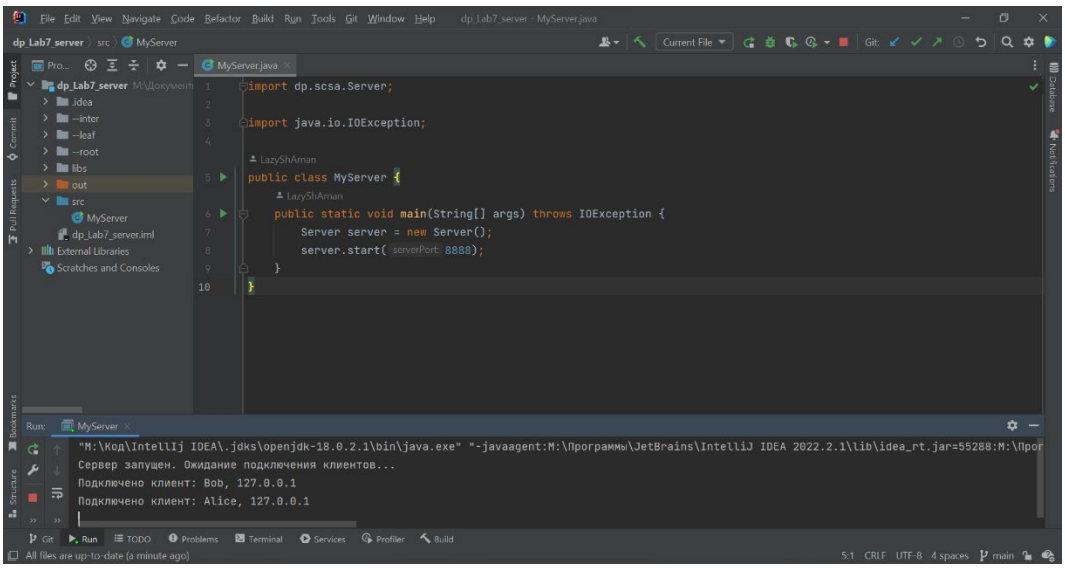

Рисунок 3 – Сервер

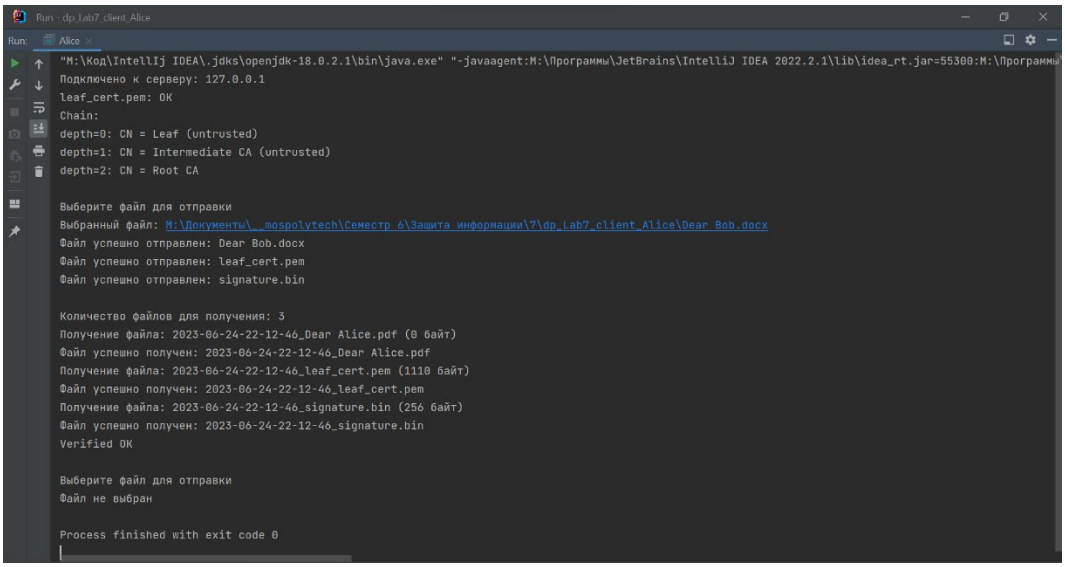

Рисунок 4 – Алиса

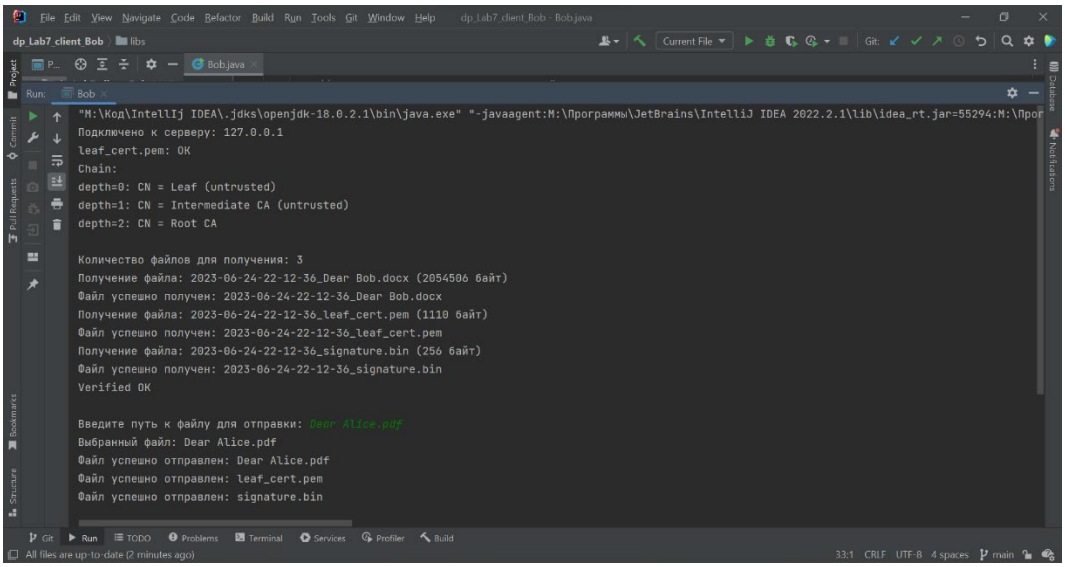

Рисунок 5 – Боб

Команды `OpenSSL`, использованные в лабораторной работе:

```
### ROOT
+ openssl genpkey -algorithm RSA -out root_keypair.pem
+ openssl req -new -subj "/CN=ROOT CA" -addext "basicConstraints=critical,CA:TRUE" -key
root_keypair.pem -out root_csr.pem
+ openssl x509 -req -in root_csr.pem -signkey root_keypair.pem -days 3650 -out
root_cert.pem

### INTERMEDIATE
+ openssl genpkey -algorithm RSA -out intermediate_keypair.pem
+ openssl req -new -subj "/CN=INTERMEDIATE CA" -addext "basicConstraints=critical,CA:TRUE"
-key intermediate_keypair.pem -out intermediate_csr.pem
+ openssl x509 -req -in intermediate_csr.pem -copy_extensions copyall -CA root_cert.pem -
CAkey root_keypair.pem -days 3650 -out intermediate_cert.pem

### LEAF
+ openssl genpkey -algorithm RSA -out leaf_keypair.pem
+ openssl req -new -subj "/CN=LEAF" -addext "basicConstraints=critical,CA:FALSE" -key
leaf_keypair.pem -out leaf_csr.pem
+ openssl x509 -req -in leaf_csr.pem -copy_extensions copyall -CA intermediate_cert.pem -
CAkey intermediate_keypair.pem -days 3650 -out leaf_cert.pem

### VERIFY CHAIN
+ openssl verify -verbose -show_chain -trusted root_cert.pem -untrusted
intermediate_cert.pem leaf_cert.pem

### GENERATE SIGNATURE
+ openssl dgst -sha512 -sign leaf_keypair.pem -out signature.bin file.txt

### PUBLIC KEY
+ openssl x509 -in leaf_cert.pem -pubkey -noout -out public_key.pem

### VERIFY SIGNATURE
+ openssl dgst -sha512 -verify public_key.pem -signature signature.bin file.txt
```

Ссылка на проект в репозитории GitHub:

– https://github.com/LazyShAman/dp/tree/main/7.