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 №4

на тему: *«Создание и верификация цифровой подписи»*

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

Слепая подпись *(или blind signature)* – это криптографический протокол, который позволяет получателю подписать сообщение, не имея полной информации о содержании сообщения. Он обеспечивает анонимность отправителя, поскольку получатель не знает, какое сообщение он подписывает, а отправитель не может узнать содержание подписанного сообщения.

Первая реализация слепых подписей была осуществлена Чаумом с помощью криптосистемы RSA:

Допустим, что изначально у Боба есть открытый ключ (p, e) , где p – это модуль, а e – публичная экспонента ключа.

1. Алиса выбирает случайный маскирующий множитель r , взаимно простой с p, и вычисляет $m' = mr^e \bmod p$.
2. Алиса посылает m' по открытому каналу Бобу.
3. Боб вычисляет $s' \equiv (m')^d \bmod p$, используя свой закрытый ключ (p, d).
4. Боб отсылает s' обратно Алисе.
5. Алиса убирает свою изначальную маскировку и получает подписанное Бобом исходное сообщение m следующим образом:

   $s' \equiv s' \times r^{-1} \bmod p \equiv m^d \bmod p$.
6. Для проверки подписи Алисе необходимо возвести подписанное Бобом сообщение в степень e. Если полученное сообщение совпадает с тем, что она отправила, подпись корректна.

Приведу в качестве примера системы слепой подписи электронное голосование.

Избиратель, используя свое устройство и подключение к интернету, заполняет электронный бюллетень, в котором указывает свой голос. Избирателю важно, чтобы избирательный пункт подтвердил его участие в голосовании и записал голос, но не знал, за кого он проголосовал.

Для достижения этой цели, на стороне клиента (избирателя) получаются данные с сервера избирательного пункта, а также генерируется случайное число. Путем выполнения специальных математических операций над переданными и сгенерированными данными информация, содержащаяся в бюллетене, становится нечитаемой для избирательного пункта.

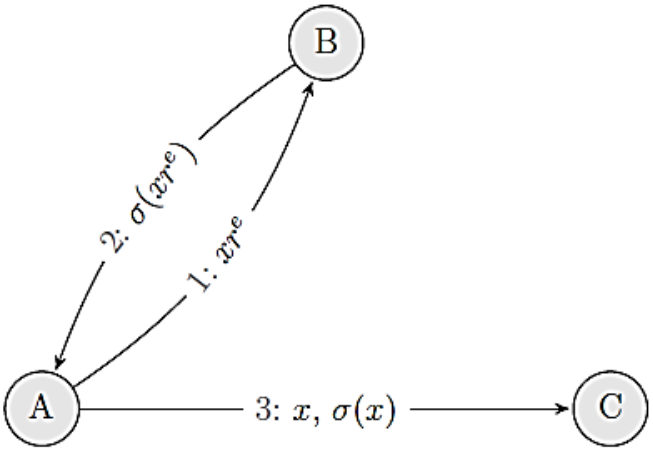
Затем избирательный пункт выполняет цифровую подпись переданного бюллетеня и отправляет его обратно на клиентскую сторону. На стороне избирателя снова применяются специальные математические операции, чтобы информация из бюллетеня стала исходной (читаемой), и подпись на бюллетене верифицируется.

В случае успешной верификации, голос засчитывается счетчиком голосов.

Этот процесс позволяет достичь двух важных целей: обеспечить конфиденциальность выбора каждого избирателя и гарантировать целостность данных в процессе передачи.

Практическая часть

Для реализации простого клиент-серверного приложения, позволяющего аккумулировать короткие анонимные сообщения согласно схеме (рисунок 1), была использована утилита `openssl`.



Здесь: A – пользователь (избиратель), B – регистратор, C – счетчик, x – сообщение (голос), r – известное только участнику A случайное число, (e, n) – открытый ключ банка. Пренебрегите реализацией правильных механизмов распределения, хранения и сертификации ключей.

Рисунок 1 – Схема работы клиент-серверного приложения

Было создано 2 проекта на языке Java: один отвечающий за клиента, другой – за сервер.

Общий принцип работы программы:

1. Создается сервер, генерируется закрытый и открытый ключи, из которых выделяются компоненты (экспонента закрытого ключа, модуль и экспонента открытого ключа).
2. Сервер запускается и ожидает подключения клиента.
3. Создается клиент и его секретное сообщение, подключается к сокету сервера.
4. Сервер передает компоненты открытого ключа клиенту.

5. Клиент получает компоненты, затемняет с их помощью свое секретное сообщение и передает на сервер.
6. Сервер получает затемненное сообщение клиента и генерирует для него затемненную подпись, а затем передает ее на клиент.
7. Клиент получает затемненную подпись с сервера и снимает с нее затемнение.
8. Подпись верифицируется и в консоль клиента выводится сообщение о том, будет ли секретное сообщение передано счетчику.

```
private static final int PORT = 8888;
private static BigInteger publicModulus;
private static BigInteger publicExponent;
private static BigInteger privateExponent;
private static BigInteger blindedSignature;
private static BigInteger blindedMessage;

public RegistrarBServer() throws IOException, NoSuchAlgorithmException,
InvalidKeySpecException {
    // Генерация закрытого и открытого ключей
    generateKeys();
}

public static void main(String[] args) throws IOException, NoSuchAlgorithmException,
InvalidKeySpecException, ClassNotFoundException {
    RegistrarBServer server = new RegistrarBServer();
    Socket socket = server.start(PORT);
    server.sendComponents(socket);
    server.receiveBlindedMessage(socket);
    blindedSignature = server.createBlindedSignature(privateExponent, publicModulus,
                                                     blindedMessage);
    server.sendBlindedSignature(socket);
}
```

Листинг 1 – RegistrarBServer: Метод main и инициализация переменных

```
private static final int SERVER_PORT = 8888;
private static final String IP_ADDRESS = "localhost";
static BigInteger publicModulus;
static BigInteger publicExponent;
static BigInteger blindedSignature;
byte[] message;
BigInteger unblindedSignature;

public static void main(String[] args) {
    VoterAClient client = new VoterAClient();
    client.message = "This is my secret vote. Shh!".getBytes();
    client.connectToServer(IP_ADDRESS, SERVER_PORT);
}
```

Листинг 2 – VoterAClient: Метод main и инициализация переменных

Методы `generateKeys()`, `getFormattedPrivateKey(byte[] privateKey)` и `getFormattedPublicKey(byte[] publicKey)` описывают генерацию ключей сервера и получение из них компонентов.

```
private static void generateKeys() throws IOException, NoSuchAlgorithmException,
InvalidKeySpecException {
    // Генерация закрытого ключа с помощью OpenSSL
    ProcessBuilder privateKeyBuilder = new ProcessBuilder("openssl", "genpkey",
            "-algorithm", "RSA", "-out", "privatekey.pem", "-pkeyopt",
            "rsa_keygen_bits:1024");
    executeCommand(privateKeyBuilder);

    // Извлечение открытого ключа из закрытого ключа с помощью OpenSSL
    ProcessBuilder publicKeyBuilder = new ProcessBuilder("openssl", "rsa",
            "-pubout", "-in", "privatekey.pem", "-out", "publickey.pem");
    executeCommand(publicKeyBuilder);

    PrivateKey privateKey =
        getFormattedPrivateKey(Files.readAllBytes(Paths.get("privatekey.pem")));
    PublicKey publicKey =
        getFormattedPublicKey(Files.readAllBytes(Paths.get("publickey.pem")));
    RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) privateKey;
    RSAPublicKey rsaPublicKey = (RSAPublicKey) publicKey;

    privateExponent = rsaPrivateKey.getPrivateExponent();
    publicModulus = rsaPublicKey.getModulus();
    publicExponent = rsaPublicKey.getPublicExponent();
}

private static PrivateKey getFormattedPrivateKey(byte[] privateKey) throws
InvalidKeySpecException, NoSuchAlgorithmException {
    String privateKeyString = new String(privateKey, StandardCharsets.UTF_8);
    String privateKeyContent = privateKeyString
            .replace("-----BEGIN PRIVATE KEY-----", "")
            .replace("-----END PRIVATE KEY-----", "")
            .replaceAll("\\s", "");

    byte[] privateKeyBytes = Base64.getDecoder().decode(privateKeyContent);
    PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(privateKeyBytes);
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");

    return keyFactory.generatePrivate(keySpec);
}

private static PublicKey getFormattedPublicKey(byte[] publicKey) throws
InvalidKeySpecException, NoSuchAlgorithmException {
    String publicKeyString = new String(publicKey, StandardCharsets.UTF_8);
    String publicKeyContent = publicKeyString
            .replace("-----BEGIN PUBLIC KEY-----", "")
            .replace("-----END PUBLIC KEY-----", "")
            .replaceAll("\\s", "");

    byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyContent);
    X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKeyBytes);
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");

    return keyFactory.generatePublic(keySpec);
}
```

Листинг 3 – RegistrarBServer: генерация ключей сервера и получение из них компонентов

Методы `start(int serverPort)`, `sendComponents(Socket socket)`, `receiveBlindedMessage(Socket socket)` и `sendBlindedSignature(Socket socket)` описывают запуск сервера и его взаимодействие с клиентом.

```
public Socket start(int serverPort) {
    try {
        ServerSocket serverSocket = new ServerSocket(serverPort);
        System.out.println("Сервер запущен. Ожидание подключения клиента...");

        Socket socket = serverSocket.accept();
        System.out.println("Подключено клиент: " +
                                socket.getInetAddress().getHostAddress());

        return socket;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

private void sendComponents(Socket socket) throws IOException {
    // Отправка открытого ключа клиенту
    ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());
    BigInteger[] components = new BigInteger[2];
    components[0] = publicModulus;
    components[1] = publicExponent;
    outputStream.writeObject(components);
}

private void receiveBlindedMessage(Socket socket) throws IOException,
ClassNotFoundException {
    // Получение затемненного сообщения клиента
    ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());
    blindedMessage = (BigInteger) inputStream.readObject();
}

private void sendBlindedSignature(Socket socket) throws IOException {
    // Отправка подписанного затемненного сообщения клиенту
    ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());
    outputStream.writeObject(blindedSignature);
}
```

Листинг 4 – RegistrarBServer: запуск сервера и его взаимодействие с клиентом

Методы `connectToServer(String serverAddress, int serverPort)`, `sendBlindedMessage(Socket socket, BigInteger message)` и `receiveBlindedSignature(Socket socket)` описывают подключение клиента к серверу и взаимодействие с ним.

```
public void connectToServer(String serverAddress, int serverPort) {
    try {
        Socket socket = new Socket(serverAddress, serverPort);
        System.out.println("Подключено к серверу: " +
                                socket.getInetAddress().getHostAddress());

        // Получение компонентов открытого ключа сервера
        ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());
        BigInteger[] components = (BigInteger[]) inputStream.readObject();
```

```
        publicModulus = components[0];
        publicExponent = components[1];

        // Blind the message
        BigInteger[] blindedMessage = blindMessage(message, publicExponent, publicModulus);
        sendBlindedMessage(socket, blindedMessage[0]);

        receiveBlindedSignature(socket);
        unblindedSignature = unblindSignature(blindedSignature, blindedMessage[1],
                                              publicModulus);

        verifySignature();

    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}

private void sendBlindedMessage(Socket socket, BigInteger message) throws IOException {
    // Отправка затемненного сообщения серверу
    ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());
    outputStream.writeObject(message);
}

private void receiveBlindedSignature(Socket socket) throws IOException,
ClassNotFoundException {
    // Получение затемненной цифровой подписи с сервера
    ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());
    blindedSignature = (BigInteger) inputStream.readObject();
}
```

Листинг 5 – VoterAClient: подключение клиента к серверу и взаимодействие с ним

Методы `blindMessage(byte[] message, BigInteger exp, BigInteger mod)` и `generateRandomBlindingFactor(BigInteger mod)` описывают генерацию затемняющего фактора и затемненного сообщения.

```
private BigInteger[] blindMessage(byte[] message, BigInteger exp, BigInteger mod) {
    BigInteger messageInt = new BigInteger(1, message);

    // Генерирует затемняющий фактор (случайное число)
    BigInteger r = generateRandomBlindingFactor(mod);
    // Blinded message = (message * r^e) mod n
    BigInteger blindedMessageInt = messageInt.multiply(r.modPow(exp, mod)).mod(mod);

    // Возвращает затемненное сообщение и затемняющий фактор
    return new BigInteger[]{blindedMessageInt, r};
}

private BigInteger generateRandomBlindingFactor(BigInteger mod) {
    BigInteger rand;
    do {
        rand = new BigInteger(mod.bitLength(), new java.security.SecureRandom());
    } while (rand.compareTo(BigInteger.ZERO) <= 0 || rand.compareTo(mod) >= 0 ||
!rand.gcd(mod).equals(BigInteger.ONE));
    return rand;
}
```

Листинг 6 – VoterAClient: генерация затемняющего фактора и затемненного сообщения

Метод `createBlindedSignature(BigInteger privateExp, BigInteger privateMod, BigInteger blindMessage)` описывает создание затемненной подписи.

```
private BigInteger createBlindedSignature(BigInteger privateExp, BigInteger privateMod,
BigInteger blindMessage) {
    return blindMessage.modPow(privateExp, privateMod);
}
```

Листинг 7 – RegistrarBServer: создание затемненной подписи

Методы `unblindSignature(BigInteger blindedSignature, BigInteger r, BigInteger modulus)` и `verifySignature()` описывают снятие затемнения с подписи и ее дальнейшую верификацию.

```
private BigInteger unblindSignature(BigInteger blindedSignature, BigInteger r, BigInteger
modulus) {
    // Unblinded message = (blinded message * r^-e) mod n
    return blindedSignature.multiply(r.modPow(new BigInteger("-1"), modulus)).mod(modulus);
}

private void verifySignature() {
    BigInteger messageInt = new BigInteger(1, message);
    System.out.println("Первоначальный  голос избирателя (в байтах) " + messageInt);
    System.out.println("Восстановленный голос избирателя (в байтах) " +
unblindedSignature.modPow(publicExponent, publicModulus).mod(publicModulus));

    if (unblindedSignature.modPow(publicExponent,
publicModulus).mod(publicModulus).equals(messageInt))
        System.out.println("Голос будет зачтен счетчиком голосов.");
    else
        System.out.println("Голос отклонен. Подпись неверна.");
}
```

Листинг 8 – VoterAClient: снятие затемнения с подписи и ее дальнейшая верификация

```
private static void executeCommand(ProcessBuilder processBuilder) throws IOException {
    processBuilder.redirectErrorStream(true);
    Process process = processBuilder.start();
    try {
        process.waitFor();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Листинг 9 – RegistrarBServer: метод, выполняющий команды openssl

Ссылка на проект в репозитории GitHub:

– https://github.com/LazyShAman/dp/tree/main/4.