ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ
ПО ДИСЦИПЛИНЕ
«Защита информации»

Руководители _____ Е.А. Харченко

Москва 2023

СПИСОК ИСПОЛНИТЕЛЕЙ

Студ. группы 201-361 _____ А.Е. Сильченко

Ход работы:

1. Генерируем открытый и приватный ключ рисунок 1.

```
//Создаем открытый и закрытый ключ
Process genpkeyProcess = Runtime.getRuntime().exec( command: "openssl genpkey -algorithm RSA -out privatekey.pem -pkeyopt rsa_keygen_bits:1024");
genpkeyProcess.waitFor();
Process rsaProcess = Runtime.getRuntime().exec( command: "openssl rsa -pubout -in privatekey.pem -out publickey.pem");
rsaProcess.waitFor();
```

Рисунок 1 – Генерация ключей.

2. Получаем модуль используя команду OpenSSL “openssl rsa -in privatekey.pem -noout -modulus” рисунок 2. Получаем публичную экспоненту рисунок 3.

```
//Получаем модуль
Process modul_exp = Runtime.getRuntime().exec( command: "openssl rsa -in privatekey.pem -noout -modulus");
modul_exp.waitFor();
InputStream m = modul_exp.getInputStream();
BufferedReader br_m = new BufferedReader(new InputStreamReader(m));
String line, modulus = "";
while ((line = br_m.readLine()) != null){
    modulus += line.trim();
    //System.out.println(line);
}
modulus = modulus.split( regex: "=")[1];
byte[] modulus_b = hexStringToByteArray(modulus);
```

Рисунок 2 – Получаем модуль.

```
String[] command = { "openssl", "rsa", "-in", "privatekey.pem", "-noout", "-text" };
ProcessBuilder pb = new ProcessBuilder(command);
pb.redirectErrorStream(true);
Process process = pb.start();
BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
String line_1;
String publicExponent = "";
while ((line_1 = reader.readLine()) != null) {
    // Поиск строки, содержащей "publicExponent"
    if (line_1.contains("publicExponent")) {
        // Разбиение строки на отдельные части
        String[] parts = line_1.trim().split( regex: "\\s+");
        // Поиск значения после "publicExponent"
        publicExponent = parts[1];
        break;
    }
}
reader.close();
// Запись результата в файл
File file = new File( pathname: "public_exponent_int.txt");
BufferedWriter writer = new BufferedWriter(new FileWriter(file));
writer.write(publicExponent);
writer.close();

BufferedReader br = new BufferedReader(new FileReader( fileName: "public_exponent_int.txt"));
String exp = br.readLine();
br.close();
exp = exp.replaceAll( regex: "[^\\d]",  replacement: "");
byte[] exp_b = exp.getBytes();
```

Рисунок 3 – Получение публичной экспоненты.

3. На клиенте генерируем текстовый файл исходя из голоса пользователя рисунок 4.

```java
//процесс голосования
System.out.println("Выберите кандидата, за которого вы хотите проголосовать: ");
System.out.println("1 - Кандидат 1");
System.out.println("2 - Кандидат 2");
System.out.println("3 - Кандидат 3");
Scanner scanner = new Scanner(System.in);
String vibor = scanner.nextLine();
if (vibor.equals("1")) {
    message = "Кандидат 1";
    FileWriter writer = new FileWriter( fileName: "message.txt");
    writer.write(message);
    writer.close();
} else if (vibor.equals("2")) {
    message = "Кандидат 2";
    FileWriter writer = new FileWriter( fileName: "message.txt");
    writer.write(message);
    writer.close();
} else if (vibor.equals("3")) {
    message = "Кандидат 3";
    FileWriter writer = new FileWriter( fileName: "message.txt");
    writer.write(message);
    writer.close();
}
```

Рисунок 4 – Генерация текстового файла.

4. Генерируем маскирующий множитель r, который взаимно простой с модулем рисунок 5.

```java
public static BigInteger generate_r(BigInteger m){
    BigInteger r;
    do {
        r = new BigInteger(m.bitLength(), new SecureRandom()); // Генерация случайного числа r
    } while (r.compareTo(m) >= 0 || !r.gcd(m).equals(BigInteger.ONE)); // Проверка на взаимно простоту с модулем
    return  r;
}
```

Рисунок 5 – Генерация маскировочного множителя r.

5. Маскируем сообщение, которое было сгенерировано на шаге 3 умножая его на маскирующий множитель в степени публичной экспоненты по модулю рисунок 6. Далее мы отправляем замаскированное сообщение на сервер Б.

```
public static BigInteger blindMessage(BigInteger m, BigInteger r, BigInteger e, BigInteger n) {
    return m.multiply(r.modPow(e, n)).mod(n);
}
```

Рисунок 6 – Маскировка сообщения.

6. Получаем частную экспоненту для дальнейшей слепой подписи рисунок 7.

```
// Вывод информации о структуре сгенерированного закрытого ключа
Process process_s = Runtime.getRuntime().exec( command: "openssl rsa -in privatekey.pem -noout -text -modulus");
Scanner scanner = new Scanner(process_s.getInputStream());
StringBuilder output = new StringBuilder();
while (scanner.hasNextLine()) {
    output.append(scanner.nextLine() + "\n");
}
scanner.close();

// Извлечение частной экспоненты
String privateExponent = output.substring(output.indexOf("privateExponent:"), output.indexOf("prime1:") - 1).split( regex:
BigInteger prExp = new BigInteger( signum: 1, hexStringToByteArray(privateExponent));
```

Рисунок 7 – Получение частной экспоненты.

7. Слепая подпись файла рисунок 8. Возводим замаскированное сообщение в степень частной экспоненты по модулю. Далее отправляем подписанное сообщение клиенту.

```
public static BigInteger signBlindedMessage(BigInteger blindedMessage, BigInteger d, BigInteger n) {
    return blindedMessage.modPow(d, n);
}
```

Рисунок 8 – Слепая подпись файла.

8. Клиент принимает подписанное замаскированное сообщение и снимает маскировку, умножая подписанное замаскированное сообщение на число обратное маскирующему множителю рисунок 9.

```
public static BigInteger unblindSignature(BigInteger blindedSignature, BigInteger r, BigInteger n) {
    return blindedSignature.multiply(r.modInverse(n)).mod(n);
}
```

Рисунок 9 – Снятие маскировки.

9. Подписанное сообщение без маскировки, оригинальное сообщение, модуль и публичная экспонента отправляются на сервер С для проверки подписи. Метод для проверки подписи представлен на рисунке 10. Для проверки подписи мы возводим подписанное сообщение в степень публичной экспоненты по модулю и сравниваем результат с исходным сообщением.

```
public static boolean verifySignature(BigInteger signature, BigInteger m, BigInteger e, BigInteger n)  {
    // Проверяем подпись
    return signature.modPow(e, n).equals(m);
}
```

Рисунок 10 – Проверка подписи.

10. В случае удачной проверки происходит подсчет голосов и результат передается на клиент рисунок 11.

```
boolean result = verifySignature(s, mes, e, mod); //проверяем подпись
if(result){
    if(v == 1){
        first++;
    }
    else if(v == 2){
        second++;
    }
    else if(v == 3){
        third++;
    }
}

//отправка данных клиенту(счетчик голосов)
DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());
out.writeInt(first);
out.writeInt(second);
out.writeInt(third);
```

Рисунок 11 – Отправка подчитанных голосов клиенту.

Итог:

Результат проведенного голосования рисунок 12.

```
Выберите кандидата, за которого вы хотите проголосовать:
1 - Кандидат 1
2 - Кандидат 2
3 - Кандидат 3
3
Кол-во голосов на данные момент:
За первого кандидата: 1
За второго кандидата: 1
За третьего кондидата: 1
Продолжить голосование? 1-да, 0-нет
1
Выберите кандидата, за которого вы хотите проголосовать:
1 - Кандидат 1
2 - Кандидат 2
3 - Кандидат 3
1
Кол-во голосов на данные момент:
За первого кандидата: 2
За второго кандидата: 1
За третьего кондидата: 1
Продолжить голосование? 1-да, 0-нет
0
Голосование завершено! Результаты:
За первого кандидата: 2
За второго кандидата: 1
За третьего кондидата: 1
```

Рисунок 12 – Результат проведенного голосования.