МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Протоколы открытого распределения ключей

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Гельфанова Даниила Руслановича

Преподаватель
аспирант _____ Р. А. Фарахутдинов
подпись, дата

Саратов 2023

1 Постановка задачи

Необходимо реализовать протокол открытого распределения ключей Диффи-Хеллмана.

2 Теоретические сведения

Определение. Если p – простое число и $g < p$, то g называется *генератором* (*примитивом*) по модулю p, если для каждого числа b от 1 до $p - 1$ существует некоторое число a: $g^a \equiv b (\mathrm{mod}\ p)$.

Также существует следующее утверждение касательно примитива по модулю p. Для того, чтобы число g являлось примитивом по модулю p, нужно, чтобы его порядок $ord(n) = p - 1$.

Определение. *Порядком* ненулевого элемента n поля F_p называется наименьшая степень t: $n^t \bmod p = 1$.

Порядок группы $F_p = p - 1$.

Теорема Лагранжа. Порядок любого элемента конечной группы является делителем порядка группы.

Алгоритм Диффи-Хеллмана – первый в истории алгоритм с открытым ключом, был изобретен в 1976 году. Его безопасность опирается на трудность вычисления дискретных логарифмов в конечном поле (в сравнении с легкостью возведения в степень в том же самом поле). Алгоритм может быть использован для распределения ключей – Алиса и Боб могут воспользоваться этим алгоритмом для генерации секретного ключа, но его нельзя использовать для шифрования и дешифрования сообщений.

Математика несложна. Сначала Алиса и Боб вместе выбирают большие числа n и g так, чтобы n было простым, а g было примитивом по модулю n. Эти 2 целых числа хранить в секрете необязательно, Алиса и Боб могут договориться об использовании по несекретному каналу. Эти числа даже могут совместно использоваться группой пользователей.

Алгоритм:

Вход: целое простое число n.

Выход: k – секретный ключ.

Шаг 1. Случайно выбирается примитивный элемент g, который является примитивом по модулю n.

Шаг 2. Алиса случайно выбирает большое целое число x.

Шаг 3. Алиса вычисляет $X = g^x \bmod n$ и пересылает X Бобу.

Шаг 4. Боб случайно выбирает большое целое число y.

Шаг 5. Боб вычисляет $Y = g^y \bmod n$ и пересылает Y Алисе.

Шаг 6. Алиса вычисляет $k = Y^x \bmod n$.

Шаг 7. Боб вычисляет $k' = X^y \bmod n$.

И k, и k' равны $g^{xy} \bmod n$. Никто из подслушивающих этот канал не сможет вычислить это значение, им известно только n, g, X, Y. Пока они не смогут вычислить дискретный логарифм и раскрыть x или y, они не смогут решить проблему. Поэтому k – это секретный ключ, который Алиса и Боб вычисляют независимо.

Алгоритм Диффи-Хеллмана имеет сложность, которая зависит от размера используемой модульной группы (числа n в алгоритме). Обычно сложность оценивается как $O(\log n)$, где p – это модуль, который используется для вычислений.

Однако важно отметить, что сложность алгоритма Диффи-Хеллмана связана с вычислениями в конечных полях и может изменяться в зависимости от конкретной реализации и параметров. Увеличение размера модульной группы может повысить уровень безопасности, но также увеличить вычислительную сложность.

3 Результаты работы

3.1 Сведения о программе

Программа была реализована на языке программирования Java. В ней есть 2 класса: $Main$ и $Participant$.

Класс $Participant$ – класс участника протокола. Для инициализации объекта $Participant$ указывается имя участника, простое число n и его примитив g. В классе реализованы следующие методы:

- `private void generatePrivateInt()` – генерация случайного числа пользователя;
- `private void generatePublicInt()` – вычисление числа, которое участник протокола посылает другому участнику;
- `public void calculateKey (BigInteger otherParticipantNum)` – вычисление секретного ключа.

В исполняемом классе $Main$ для входного простого числа n случайным образом выбирается примитивный корень g. Затем создаются 2 объекта Participant – Алиса и Боб, для которых и происходит установление секретного ключа k. В этом классе описаны следующие методы:

- `public static BigInteger generatePrimitiveRoot(BigInteger n)` – генерация примитива по модулю n. В данном методы выбирается случайное число до $n-1$. И начиная от него в цикле происходит поиск примитивного корня.
- `public static boolean isPrimitive(BigInteger prime, BigInteger n, ArrayList<BigInteger> orders)` – проверка, является число примитивом по утверждению, описанному в пункте 2.
- `public static ArrayList<BigInteger> getDivisors(BigInteger num)` – метод для получения списка делителей числа.

3.2 Тестирование программы

Для реализации модульного тестирования были написаны тестовые классы $ParticipantTEST$ и $MainTEST$.

В классе $ParticipantTEST$ содержится один тестовый метод, в котором создаются 2 Participant объекта и проверяется, что сгенерированные ключи для каждого из участников одинаковы. Результат отработки теста представлен на рисунке 1.

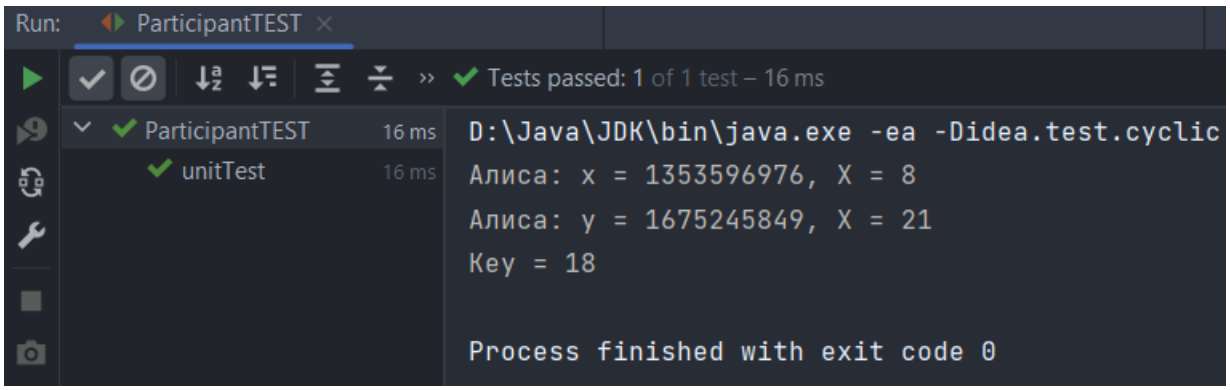

Рисунок 1 – Запуск ParticipantTEST

В классе $MainTEST$ содержится 3 тестовых метода. Первый метод проверяет корректность отработки метода $getDivisors()$ нахождения делителей числа, второй метод проверяет работу метода $isPrimitive$, а третий – проверку генерации примитивного корня $generatePrimitiveRoot$. Результат отработки тестов представлен на рисунке 2.

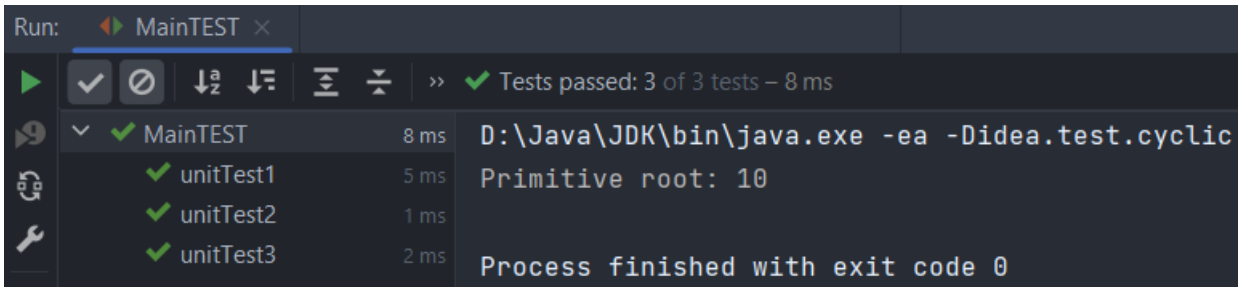

Рисунок 2 – Запуск MainTest

На рисунках 3-6 представлено тестирование программы.

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 1. Diffie-Hellman\out\artif
acts\Diffie_Hellman_jar> java -jar Diffie-Hellman.jar
Входные параметры заданы некорректно.
java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
        at Main.main(Main.java:9)
```

Рисунок 3 – Негативное тестирование (входной параметр не указан)

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 1. Diffie-Hellman\out\artif
acts\Diffie_Hellman_jar> java -jar Diffie-Hellman.jar test
Входные параметры заданы некорректно.
java.lang.NumberFormatException: For input string: "test"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatE
xception.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:668)
        at java.base/java.math.BigInteger.<init>(BigInteger.java:536)
        at java.base/java.math.BigInteger.<init>(BigInteger.java:674)
        at Main.main(Main.java:9)
```

Рисунок 4 – Негативное тестирование (входной параметр не является числом)

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 1. Diffie-Hellman\out\artif
acts\Diffie_Hellman_jar> java -jar Diffie-Hellman.jar 1024
Exception in thread "main" java.lang.IllegalArgumentException: Введенное число не
 является простым.
        at Main.main(Main.java:16)
```

Рисунок 5 – Негативное тестирование (входной параметр не является простым числом)

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 1. Diffie-Hellman\out\artif
acts\Diffie_Hellman_jar> java -jar Diffie-Hellman.jar 1021
В качестве примитивного корня выбрано g = 823
Пользователь "Алиса" сгенерировал следующее большое целое число: 831339243
Пользователь "Алиса" отослал пользователю "Боб" число X = 524

Пользователь "Боб" сгенерировал следующее большое целое число: 3094223639
Пользователь "Боб" отослал пользователю Алиса число Y = 209

Пользователь "Алиса" вычислил ключ 54
Пользователь "Боб" вычислил ключ 54
Пользователи успешно сгенерировали секретный ключ: 54
```

Рисунок 6 – Положительное тестирование (входной параметр 1021)

На рисунке 5 представлено тестирование работы программы для решения системы линейных уравнений над конечным полем.

ПРИЛОЖЕНИЕ А

Листинг программы

```
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        BigInteger n;
        try {
            n = new BigInteger(args[0]);
        } catch (Exception e){
            System.out.println("Входные параметры заданы
некорректно.");
            e.printStackTrace();
            return;
        }
        if (!n.isProbablePrime(100)){
            throw new IllegalArgumentException("Введенное число не
является простым.");
        }
        BigInteger g = generatePrimitiveRoot(n);
        System.out.println("В качестве примитивного корня выбрано g = "
+ g);
        Participant alice = new Participant("Алиса", n, g);
        Participant bob = new Participant("Боб", n, g);
        System.out.println("Пользователь \"" + alice.getName() + "\"
сгенерировал следующее большое целое число: " + alice.getPrivateInt());
        System.out.println("Пользователь \"" + alice.getName() + "\"
отослал пользователю \"" + bob.getName() + "\" число X = " +
alice.publicInt);
        bob.calculateKey(alice.publicInt);
        System.out.println();
        System.out.println("Пользователь \"" + bob.getName() + "\"
сгенерировал следующее большое целое число: " + bob.getPrivateInt());
        System.out.println("Пользователь \"" + bob.getName() + "\"
отослал пользователю " + alice.getName() + " число Y = " +
bob.publicInt);
        alice.calculateKey(bob.publicInt);
        System.out.println();
        System.out.println("Пользователь \"" + alice.getName() + "\"
вычислил ключ " + alice.getKey());
        System.out.println("Пользователь \"" + bob.getName() + "\"
вычислил ключ " + bob.getKey());
        System.out.println("Пользователи успешно сгенерировали
секретный ключ: " + alice.getKey());
    }

    private static BigInteger generatePrimitiveRoot(BigInteger n) {
        Random rand = new Random();
        BigInteger prime = new BigInteger(n.bitLength(), rand);
        ArrayList<BigInteger> orders =
getDivisors(n.subtract(BigInteger.ONE));
        for(;;){
            if (prime.compareTo(n) >= 0){
```

```
                prime = BigInteger.ONE;
            }
            if (isPrimitive(prime, n, orders)){
                return prime;
            }
            prime = prime.add(BigInteger.ONE);
        }
    }

    private static boolean isPrimitive(BigInteger prime, BigInteger n,
ArrayList<BigInteger> orders) {
        if (prime.compareTo(BigInteger.ONE) < 0 || prime.compareTo(n)
>= 0) {
            return (false);
        }
        for (BigInteger each : orders) {
            if (prime.modPow(each, n).compareTo(BigInteger.ONE) == 0) {
                if (each.compareTo(n.subtract(BigInteger.ONE)) == 0) {
                    return (true);
                }
                break;
            }
        }
        return (false);
    }

    private static ArrayList<BigInteger> getDivisors(BigInteger num) {
        ArrayList<BigInteger> divisors = new ArrayList<>();
        for        (BigInteger        i        =        BigInteger.ONE;
i.compareTo(num.divide(BigInteger.TWO).add(BigInteger.ONE)) < 0; i =
i.add(BigInteger.ONE)) {
            if (num.mod(i).compareTo(BigInteger.ZERO) == 0){
                divisors.add(i);
            }
        }
        divisors.add(num);
        return (divisors);
    }
}

import java.math.BigInteger;
import java.util.Random;

public class Participant {
    private String name;
    private BigInteger n, g, key;
    private BigInteger privateInt; // случайное большое целое число
    public BigInteger publicInt; // это число пересылается другому
пользователю

    public BigInteger getPrivateInt() {
        return privateInt;
    }

    public BigInteger getKey(){
        return key;
    }
```

```
    public String getName() {
        return name;
    }

    public Participant(String name, BigInteger n, BigInteger g){
        try {
            this.name = name;
            this.n = n;
            this.g = g;
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }
        this.generatePrivateInt();
        this.generatePublicInt();
    }

    private void generatePrivateInt() {
        Random rand = new Random();
        this.privateInt = new BigInteger(32, rand); // от 0 до 2^32 - 1
    }

    private void generatePublicInt() {
        this.publicInt = this.g.modPow(this.privateInt, n);
    }

    public void calculateKey (BigInteger otherParticipantNum){
        this.key = otherParticipantNum.modPow(this.privateInt, this.n);
    }
}


import org.junit.Assert;
import org.junit.Test;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;

public class MainTEST {
    @Test
    public void unitTest1() {
        ArrayList<BigInteger> expectedDivisors = new ArrayList<>(
                List.of(new BigInteger("1"),
                        new BigInteger("2"),
                        new BigInteger("11"),
                        new BigInteger("22")));
        ArrayList<BigInteger>  actualDivisors  =  Main.getDivisors(new
BigInteger("22"));
        Assert.assertEquals(actualDivisors, expectedDivisors);
    }

    @Test
    public void unitTest2() {
        boolean actualResult = Main.isPrimitive(new BigInteger("5"), new
BigInteger("23"), Main.getDivisors(new BigInteger("22")));
        Assert.assertTrue(actualResult);
        actualResult  =  Main.isPrimitive(new  BigInteger("6"),  new
BigInteger("23"), Main.getDivisors(new BigInteger("22")));
```

```
        Assert.assertFalse(actualResult);
    }

    @Test
    public void unitTest3() {
        BigInteger primitiveRoot = Main.generatePrimitiveRoot(new
BigInteger("23"));
        System.out.println("Primitive root: " + primitiveRoot);
        Assert.assertTrue(Main.isPrimitive(primitiveRoot, new
BigInteger("23"), Main.getDivisors(new BigInteger("22"))));
    }
}


import org.junit.Assert;
import org.junit.Test;
import java.math.BigInteger;

public class ParticipantTEST {
    @Test
    public void unitTest(){
        Participant alice = new Participant("Алиса", new
BigInteger("23"), new BigInteger("5"));
        Participant bob = new Participant("Алиса", new BigInteger("23"),
new BigInteger("5"));
        System.out.println(alice.getName() + ": x = " +
alice.getPrivateInt() + ", X = " + alice.publicInt);
        System.out.println(bob.getName() + ": y = " +
bob.getPrivateInt() + ", X = " + bob.publicInt);
        alice.calculateKey(bob.publicInt);
        bob.calculateKey(alice.publicInt);
        System.out.println("Key = " + alice.getKey());
        Assert.assertEquals(alice.getKey(), bob.getKey());
    }
}
```