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

**Схемы аутентификации**

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 схему идентификации Гиллу-Кискате.

**2 Теоретические сведения**

Протокол Фейге-Фиата-Шамира был первым, обеспечивающим практическую идентификацию. Этот протокол минимизировал вычисления, увеличивая число итераций и аккредитаций на итерацию. Для некоторых реализаций, например, для смарт-карт такой способ вычислений не слишком подходит. Обмены с внешним миром требуют затрат времени, а хранение данных для каждой аккредитации может быстро исчерпать ограниченные возможности карточки.

Луи Гиллу и Жан-Жак Кискате разработали алгоритм идентификации с нулевым разглашением, более подходящий для подобных приложений. Обмены информацией между Пегги и Виктором, а также параллельные аккредитации в каждом обмене сведены к абсолютному минимуму: для каждого доказательства существует только один обмен, в котором выполняется только одна аккредитация. Чтобы достичь при использовании схемы Гиллу-Кискате того же уровня безопасности, что и в схеме Фейге-Фиата-Шамира, потребуется выполнить в три раза больше вычислений. Так же, как и с в случае схемы Фейге-Фиата-Шамира, этот алгоритм идентификации можно превратить в алгоритм цифровой подписи.

Пусть Пегги – это смарт-карта, которая собирается доказать свою подлинность Виктору. Идентификация Пегги проводится по ряду атрибутов, представляющих собой строку данных, содержащих название карточки, период действия, номер банковского счета и другие, подтверждаемые ее применимость, данные. Эта битовая строка называется . Эта строка данных аналогична открытому ключу. Другой открытой информацией, общей для всех «Пегги», которые могут использовать это приложение, является показатель степени и модуль , где – это произведение дух хранящихся в секрете простых чисел. Показатель степени выбирается так, чтобы . Закрытым ключом служит значение , рассчитываемое так, чтобы .

Пегги посылает Виктору свои атрибуты . Далее Пегги хочет доказать Виктору, что это именно ее атрибуты. Для этого она должна убедить Виктора, что ей известно значение . Вот такой протокол она для этого использует.

Алгоритм работы схемы идентификации Гиллу-Кискате:

Вход: открытый ключ Пегги: .

Выход: Доказательство подлинности Пегги либо опровержение подлинности.

Шаг 1. Пегги выбирает случайное целое , находящееся в диапазоне от до . Она вычисляет и отправляет его Виктору.

Шаг 2. Виктор выбирает случайное целое , находящееся в диаразоне от 0 до . Он посылает Пегги.

Шаг 3. Пегги вычисляет и посылает его Виктору.

Шаг 4. Виктор вычисляет . Если , то подлинность Пегги доказана.

Используемая здесь математика не слишком сложна:

Так как по определению удовлетворяет:

**3 Результаты работы**

**3.1 Сведения о программе**

Программа была реализована на языке программирования Java. В ней есть 4 класса: , , и .

В классе происходит считывание входных параметров: числа и двух простых чисел и . Затем случайным образом генерируется показатель степени .

Класс – класс реализации самого протокола. Для инициализации передаются следующие параметры: . Создаются объекты классов *Peggy* и . После чего происходит запуск работы протокола. Каждому шагу алгоритма выше соответствует собственный метод.

Класс – класс участника Пегги. Для создания объекта этого класса нужно передать . При инициализации объекта данного класса генерируется случайное число и ищется значение числа . В классе описаны следующие методы:

* public String toString()– вывод информации о Пегги (открытый и закрытый ключи).
* public BigInteger calculated() – вычисление числа согласно алгоритму выше.
* private void setB() – вычисление и установка числа .

Класс – класс проверяющей стороны (Виктор). Пегги. Для создания объекта этого класса нужно передать показатель степени . При инициализации объекта данного класса генерируется случайное число . В классе описан следующий метод:

* public void calculateT\_(Peggy peggy) – вычисление числа согласно алгоритму выше.

**3.2 Тестирование программы**

Для реализации модульного тестирования были написаны тестовые классы и .

В классе содержится 2 метода для тестирования самого протокола: один негативный и один положительный. В негативном тесте проверяется случай вызова исключения, когда Пегги не подтвердила свою подлинность. В положительном – подтверждение подлинности Пегги. Результат отработки теста представлен на рисунке 1.

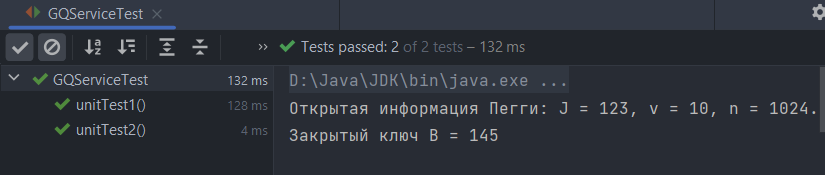


Рисунок 1 – Запуск

В классе содержится 2 тестовых метода: проверка вывода информации о Пегги и проверка того, что число вычисляется корректно. Результат отработки тестов представлен на рисунке 2.

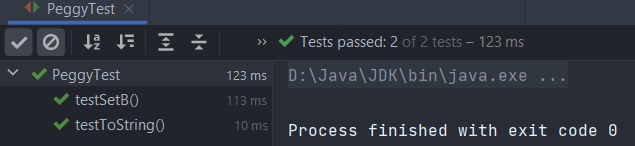


Рисунок 2 – Запуск

На рисунках 3-5 представлено тестирование программы.



Рисунок 3 – Негативное тестирование (входные параметры не указаны)



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

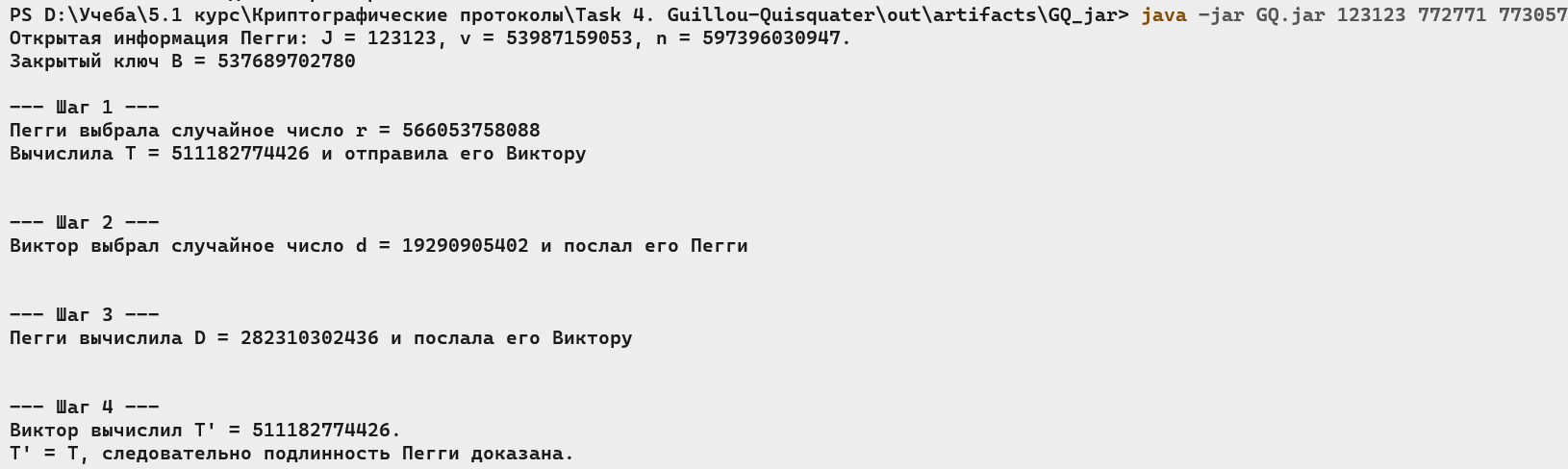


Рисунок 5 – Положительное тестирование (входные параметры ,,

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import java.math.BigInteger;

import java.util.Random;

public class GuillouQuisquater {

public static void main(String[] args) throws IllegalAccessException {

if (args.length == 0) {

System.out.println("Входные параметры отсутсвуют");

return;

}

if (args[0].equals("/help") || args[0].equals("h")) {

System.out.println("""

Программе должны передаваться следующие параметры:

\t- Число J (открытый ключ)

\t- 2 простых числа p и q""");

return;

}

if (args.length < 3) {

System.out.println("Передано некорректное число параметров");

return;

}

if (args[1].equals(args[2])) {

System.out.println("Числа p и q должны быть различными");

return;

}

BigInteger j, v, n, phi;

try {

Random rnd = new Random();

j = new BigInteger(args[0]);

BigInteger p = new BigInteger(args[1]);

BigInteger q = new BigInteger(args[2]);

if (!p.isProbablePrime(100) || !q.isProbablePrime(100)) {

System.out.println("Числа p и q должны быть простыми.");

throw new IllegalArgumentException();

}

phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

v = new BigInteger(phi.bitLength(), rnd).mod(phi);

while (!v.gcd(phi).equals(BigInteger.ONE)) {

v = v.add(BigInteger.ONE).mod(phi);

if (v.equals(BigInteger.ZERO)) {

v = BigInteger.ONE;

}

}

n = p.multiply(q);

} catch (IllegalArgumentException e) {

System.out.println("Ошибка в чтении входных параметров.");

return;

}

GQService service = new GQService(j, v, n, phi);

}

}

import java.math.BigInteger;

public class GQService {

public Peggy peggy;

public Victor victor;

public GQService(BigInteger j, BigInteger v, BigInteger n, BigInteger phi) throws IllegalAccessException {

this.peggy = new Peggy(j, v, n, phi);

System.out.println(peggy);

this.victor = new Victor(v);

this.step1();

}

private void step1() throws IllegalAccessException {

BigInteger T = peggy.getR().modPow(peggy.getV(), peggy.getN());

victor.setT(T);

System.out.printf("""

\n--- Шаг 1 ---

Пегги выбрала случайное число r = %d

Вычислила Т = %d и отправила его Виктору

""", peggy.getR(), T

);

this.step2();

}

private void step2() throws IllegalAccessException {

peggy.setD(victor.getRandomInt());

System.out.printf("""

\n--- Шаг 2 ---

Виктор выбрал случайное число d = %d и послал его Пегги

""", victor.getRandomInt()

);

this.step3();

}

private void step3() throws IllegalAccessException {

BigInteger D = peggy.calculateD();

victor.setD(D);

System.out.printf("""

\n--- Шаг 3 ---

Пегги вычислила D = %d и послала его Виктору

""", D

);

this.step4();

}

private void step4() throws IllegalAccessException {

victor.calculateT\_(peggy);

if (!victor.getT().equals(victor.getT\_())) {

throw new IllegalAccessException("Подлинность Пегги недоказана, так как T' != T\nT = " + victor.getT() + ", T' = " + victor.getT\_());

}

System.out.printf("""

\n--- Шаг 4 ---

Виктор вычислил Т' = %d.

T' = T, следовательно подлинность Пегги доказана.

""", victor.getT\_()

);

}

public Victor getVictor() {

return victor;

}

}

import java.math.BigInteger;

import java.security.SecureRandom;

public class Peggy {

private final BigInteger v, n, phi;

private BigInteger j, b, r, d;

public Peggy(BigInteger j, BigInteger v, BigInteger n, BigInteger phi) {

this.j = j;

this.v = v;

this.n = n;

this.phi = phi;

SecureRandom rnd = new SecureRandom();

do {

r = new BigInteger(n.bitLength(), rnd).mod(n);

} while (r.equals(BigInteger.ZERO));

this.setB();

}

@Override

public String toString() {

return String.format("""

Открытая информация Пегги: J = %d, v = %d, n = %d.

Закрытый ключ B = %d""", j, v, n, b);

}

public BigInteger calculateD() {

return r.multiply(b.modPow(d, n)).mod(n);

}

public BigInteger getN() {

return n;

}

public BigInteger getJ() {

return j;

}

public BigInteger getV() {

return v;

}

public BigInteger getR() {

return r;

}

public BigInteger getB() {

return b;

}

private void setB() {

try {

this.b = j.modInverse(n).modPow(v.modInverse(phi), n);

} catch (ArithmeticException e) {

j = j.add(BigInteger.ONE);

setB();

}

}

public void setD(BigInteger d) {

this.d = d;

}

}

import java.math.BigInteger;

import java.security.SecureRandom;

public class Victor {

private BigInteger T, T\_, D;

private final BigInteger d;

public Victor(BigInteger v) {

SecureRandom rnd = new SecureRandom();

this.d = new BigInteger(v.bitLength(), rnd).mod(v);

}

public BigInteger getT() {

return T;

}

public BigInteger getT\_() {

return T\_;

}

public BigInteger getRandomInt() {

return d;

}

public void setT(BigInteger T) {

this.T = T;

}

public void setD(BigInteger D) {

this.D = D;

}

public void calculateT\_(Peggy peggy) {

this.T\_ = D.modPow(peggy.getV(), peggy.getN())

.multiply(peggy.getJ().modPow(d, peggy.getN()))

.mod(peggy.getN());

}

}

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.Test;

import java.math.BigInteger;

import static org.junit.jupiter.api.Assertions.\*;

class GQServiceTest {

@Test

public void unitTest1() {

BigInteger j = new BigInteger("123");

BigInteger v = new BigInteger("10");

BigInteger n = new BigInteger("1024");

BigInteger phi = new BigInteger("13");

java.lang.IllegalAccessException thrown = assertThrows(

java.lang.IllegalAccessException.class,

() -> new GQService(j, v, n, phi),

"Ожидалось исключение в new GQService(...), но его не было."

);

}

@Test

public void unitTest2() throws IllegalAccessException {

BigInteger j = new BigInteger("10");

BigInteger v = new BigInteger("19");

BigInteger n = new BigInteger("221");

BigInteger phi = new BigInteger("192");

GQService service = new GQService(j, v, n, phi);

Victor victor = service.getVictor();

Assertions.assertEquals(victor.getT\_(), victor.getT());

}

}

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import java.math.BigInteger;

class PeggyTest {

Peggy peggy;

@BeforeEach

void prepareData() {

peggy = new Peggy(new BigInteger("10"), new BigInteger("19"), new BigInteger("221"), new BigInteger("192"));

}

@Test

void testToString() {

Assertions.assertTrue(peggy.toString().startsWith("""

Открытая информация Пегги: J = 10, v = 19, n = 221.

Закрытый ключ B = """));

}

@Test

void testSetB() {

Assertions.assertEquals(peggy.getJ().multiply(peggy.getB().modPow(peggy.getV(), peggy.getN())).mod(peggy.getN()), BigInteger.ONE);

}

}