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

**Скрытый канал связи**

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 скрытый канал связи на основе схемы Онга-Шнорра-Шамира.

**2 Теоретические сведения**

Данный скрытый канал, разработанный Густавусом Симмонсом, использует схему идентификации Онга-Шнорра-Шамира. Как и в оригинальной схеме, отправитель (Алиса) выбирает общедоступный модуль (большое нечетное целое число) и закрытый ключ так, чтобы они были взаимно простыми числами. В отличие от оригинальной схемы, используется совместно Алисой и Бобом, получателем в скрытом канале. Открытый ключи вычисляется следующим образом:

Если Алисе нужно отправить скрытое сообщение в безобидном сообщении , она сначала проверяет, чтобы пары и , а также и являются взаимно простыми числами. Алиса вычисляет:

Пара чисел и представляет собой подпись в традиционной схеме Онга-Шнорра-Шамира и одновременно является носителем скрытого сообщения.

Надзиратель Уолтер может проверить подлинность сообщение, как это принято в схеме Онга-Шнорра-Шамира, но Боб может сделать и еще кое-что. Он может проверить подлинность сообщения (всегда возможно, что Уолтер попытается ему подсунуть поддельное сообщение). Он проверяет, что:

Если подлинность сообщения доказана, получатель может извлечь и скрытое сообщение, используя следующую формулу:

Эта схема работоспособна, но помните, что сама схема Онга-Шнорра-Шамира была взломана.

Алгоритм работы скрытого канала связи на основе схемы Онга-Шнорра-Шамира:

Вход: – большое целое нечетное число, скрытое сообщение .

Выход: «Боб получил скрытое сообщение » или «Боб выяснил, что сообщение не является подлинным».

Шаг 1. Генерируется закрытый ключ и открытый ключ

Шаг 2. Алиса выбирает открытое сообщение так, чтобы пары и , а также и являются взаимно простыми числами. Если условие не выполняется, то протокол завершен с ошибкой, нужно выбрать другое .

Шаг 3. Алиса вычисляет подпись:

и передает сообщение с подписью Уолтеру.

Шаг 4. Уолтер передает Бобу и . Если Уолтер захотел подделать сообщение, то вместо он может передать что-то другое.

Шаг 5. Боб проверяет подлинность сообщение, то есть проверяет, что Если сравнение не выполняется, то выход из алгоритма, результат: «Боб выясним, что сообщение не является подлинным».

Шаг 6. Боб вычисляет скрытое сообщение:

**3 Результаты работы**

**3.1 Сведения о программе**

Программа была реализована на языке программирования Java. В ней есть 6 классов: , , , , и .

В классе происходит считывание входных параметров: числа и скрытого сообщения .

Класс – класс реализации самого протокола. Для инициализации передаются и. При инициализации создаются объекты класса , , . Каждому шагу алгоритма выше соответствует собственный метод.

Класс – класс отправителя Алиса. Для создания объекта этого класса нужно передать . При инициализации объекта данного класса генерируется закрытый ключ и вычисляется открытый ключ . Также выбирается открытое сообщение . В классе описаны следующие методы:

- public BigInteger findCoprime(BigInteger n) – метод для нахождения взаимно простого числа для ;

- public boolean checkParameters() – проверка условия на шаге 2 алгоритма;

- public void signMessage() – вычисление подписи.

Класс – класс получателя Боба. В классе описаны 2 метода:

* public boolean checkMessage() – проверка подлинности сообщения.
* public void calculateM() – вычисление скрытого сообщения .

Класс – класс надзирателя Уолтера. В нем описан 1 метод:

- public void badWolter() – изменение переданного Алисой сообщения.

Класс – класс для подписи, в нем содержатся 2 переменные и .

**3.2 Тестирование программы**

Для реализации модульного тестирования был написан тестовый класс .

В классе содержится 3 метода для тестирования. Один из методов проверяет случай, когда не выполняется взаимная простота чисел и . Второй метод проверяет случай, когда Уолтер подделывает сообщение. Третий метод проверяет правильность нахождения взаимно простых чисел. Результат отработки теста представлен на рисунке 1.

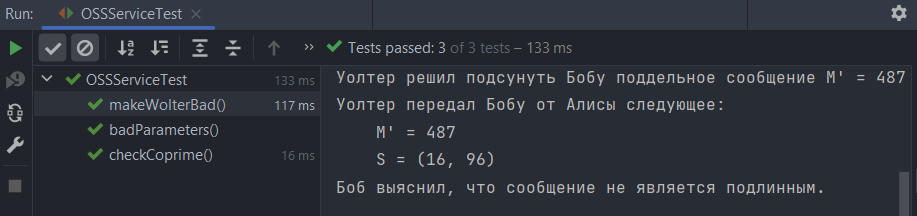


Рисунок 1 – Запуск

На рисунке 2 представлено негативное тестирование программы.

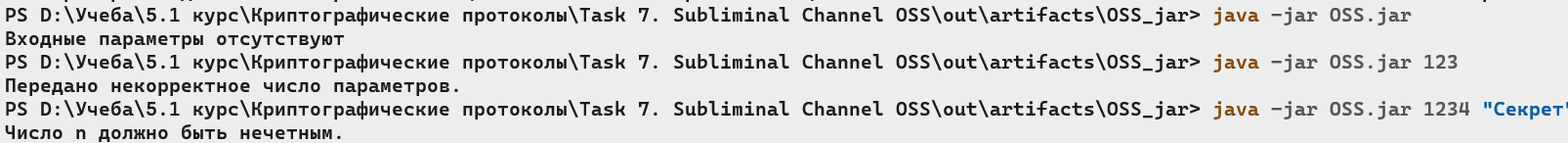


Рисунок 2 – Негативное тестирование

На рисунке 3 представлено положительное тестирование программы.

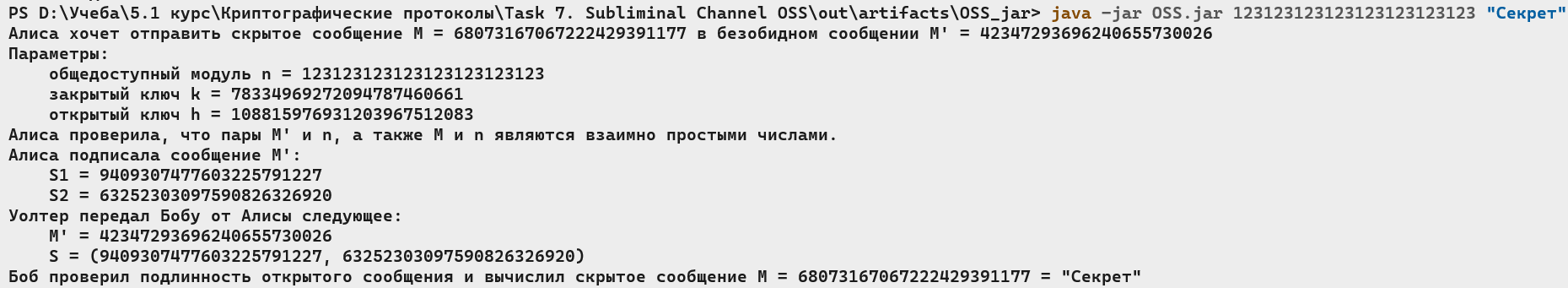


Рисунок 3 – Положительное тестирование

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import java.math.BigInteger;

public class OSSChannel {

public static void main(String[] args) {

if (args.length == 0) {

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

return;

}

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

System.out.println("""

Программе должны передаваться следующие параметры:

\t- большое целое нечетное число n

\t- скрытое сообщение M""");

return;

}

if (args.length < 2) {

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

return;

}

BigInteger n;

String secretMessage;

try {

n = new BigInteger(args[0]);

if (n.mod(BigInteger.TWO).equals(BigInteger.ZERO)){

throw new IncorrectParametersException("Число n должно быть нечетным.");

}

secretMessage = args[1];

} catch (IndexOutOfBoundsException e) {

System.out.println("Выход за пределы массива.");

return;

} catch (NumberFormatException e) {

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

return;

} catch (IncorrectParametersException e){

System.out.println(e.getMessage());

return;

}

OSSService service = new OSSService(n, secretMessage);

service.runProtocol();

}

}

import java.math.BigInteger;

import java.util.HashMap;

class IncorrectParametersException extends Exception {

public IncorrectParametersException(String s) {

super(s);

}

}

class BobSignatureException extends Exception {

public BobSignatureException(String s) {

super(s);

}

}

public class OSSService {

Alice alice;

Bob bob;

Wolter wolter;

private HashMap<BigInteger, String> intToStringMap = new HashMap<>();

private boolean exceptionHandled = false;

public OSSService(BigInteger n, String secretMessage) {

this.alice = new Alice(n);

this.intToStringMap.put(alice.getM(), secretMessage);

bob = new Bob(alice.getN(), alice.getK(), alice.getH());

wolter = new Wolter(alice.getM\_());

System.out.println(this);

}

@Override

public String toString() {

return String.format("""

Алиса хочет отправить скрытое сообщение М = %d в безобидном сообщении M' = %d

Параметры:

общедоступный модуль n = %d

закрытый ключ k = %d

открытый ключ h = %d"""

, alice.getM(), alice.getM\_(), alice.getN(), alice.getK(), alice.getH());

}

public void runProtocol() {

try {

if (alice.checkParameters()) {

System.out.println("Алиса проверила, что пары M' и n, а также M и n являются взаимно простыми числами.");

this.signMessage();

} else {

throw new IncorrectParametersException("ОШИБКА: Пары чисел M' и n, а также M и n не являются взаимно простыми числами");

}

} catch (IncorrectParametersException e) {

System.out.println(e.getMessage());

this.exceptionHandled = true;

return;

}

}

private void signMessage() {

alice.signMessage();

System.out.printf("""

Алиса подписала сообщение M':

S1 = %d

S2 = %d\n""", alice.getS().getS1(), alice.getS().getS2());

this.sendMessageWithSignature();

}

private void sendMessageWithSignature() {

bob.setM\_S(wolter.getM\_(), alice.getS());

if (!wolter.getM\_().equals(alice.getM\_())) {

System.out.println("Уолтер решил подсунуть Бобу поддельное сообщение M' = " + wolter.getM\_());

}

System.out.printf("""

Уолтер передал Бобу от Алисы следующее:

M' = %d

S = %s\n""", wolter.getM\_(), alice.getS());

try {

if (bob.checkMessage()) {

bob.calculateM();

System.out.println("Боб проверил подлинность открытого сообщения и вычислил скрытое сообщение M = " + bob.getM() + " = \"" + intToStringMap.get(bob.getM()) + "\"");

} else{

throw new BobSignatureException("Боб выяснил, что сообщение не является подлинным.");

}

} catch (BobSignatureException e) {

System.out.println(e.getMessage());

this.exceptionHandled = true;

return;

}

}

public void makeWolterBad() {

wolter.badWolter();

}

public Alice getAlice() {

return alice;

}

public boolean getException() {

return exceptionHandled;

}

}

import java.math.BigInteger;

import java.security.SecureRandom;

public class Wolter {

private BigInteger M\_;

public Wolter (BigInteger M\_) {

this.M\_ = M\_;

}

public void badWolter() {

SecureRandom rnd = new SecureRandom();

this.M\_ = new BigInteger(this.M\_.bitLength(), rnd);

}

public BigInteger getM\_() {

return M\_;

}

}

import java.math.BigInteger;

public class Bob {

private BigInteger n, k, h, M, M\_;

Signature s;

public Bob (BigInteger n, BigInteger k, BigInteger h) {

this.n = n;

this.k = k;

this.h = h;

}

public void setM\_S(BigInteger M\_, Signature s) {

this.M\_ = M\_;

this.s = s;

}

public BigInteger getM() {

return M;

}

public boolean checkMessage() {

BigInteger s1 = s.getS1();

BigInteger s2 = s.getS2();

BigInteger check = s1.multiply(s1).subtract(s2.multiply(s2).multiply(k.multiply(k).modInverse(n))).mod(n);

return check.equals(M\_.mod(n));

}

public void calculateM() {

BigInteger div = s.getS1().add(s.getS2().multiply(k.modInverse(n))).modInverse(n);

this.M = M\_.multiply(div).mod(n);

}

}

import java.math.BigInteger;

import java.security.SecureRandom;

public class Alice {

private BigInteger n, k, h, M, M\_;

private Signature s;

public Alice(BigInteger n) {

this.n = n;

this.M = this.findCoprime(this.n);

this.k = this.findCoprime(this.n);

this.h = this.k.multiply(this.k).negate().mod(n);

this.M\_ = this.findCoprime(this.n);

}

public BigInteger getN() {

return n;

}

public BigInteger getK() {

return k;

}

public BigInteger getH() {

return h;

}

public BigInteger getM() {

return M;

}

public BigInteger getM\_() {

return M\_;

}

public Signature getS() {

return s;

}

public void setM(BigInteger m) {

M = m;

}

public void setM\_(BigInteger m\_) {

M\_ = m\_;

}

public BigInteger findCoprime(BigInteger n) {

SecureRandom rnd = new SecureRandom();

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

while(!coprime.gcd(n).equals(BigInteger.ONE)) {

coprime = coprime.add(BigInteger.ONE).mod(n);

}

return coprime;

}

public boolean checkParameters() {

return M\_.gcd(n).equals(BigInteger.ONE) && M.gcd(n).equals(BigInteger.ONE);

}

public void signMessage() {

BigInteger invTwo = BigInteger.TWO.modInverse(n);

BigInteger invM = M.modInverse(n);

this.s = new Signature(

invTwo.multiply(M\_.multiply(invM).add(M)).mod(n),

k.multiply(invTwo).multiply(M\_.multiply(invM).subtract(M)).mod(n));

}

}

import java.math.BigInteger;

public class Signature {

private final BigInteger S1, S2;

public Signature(BigInteger S1, BigInteger S2) {

this.S1 = S1;

this.S2 = S2;

}

public BigInteger getS1() {

return S1;

}

public BigInteger getS2() {

return S2;

}

@Override

public String toString() {

return "(" + S1 + ", " + S2 + ")";

}

}

import java.math.BigInteger;

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

class OSSServiceTest {

@org.junit.jupiter.api.Test

void badParameters() {

OSSService service = new OSSService(new BigInteger("1024"), "Test");

service.getAlice().setM(new BigInteger("256"));

service.getAlice().setM\_(new BigInteger("1024"));

service.runProtocol();

assertTrue(service.getException());

}

@org.junit.jupiter.api.Test

void makeWolterBad() {

OSSService service = new OSSService(new BigInteger("1023"), "Test");

service.makeWolterBad();

service.runProtocol();

assertTrue(service.getException());

}

@org.junit.jupiter.api.Test

void checkCoprime() {

Alice alice = new Alice(new BigInteger("1023"));

BigInteger n = alice.getN();

BigInteger a = new BigInteger("100");

assertTrue(alice.getM\_().gcd(n).equals(BigInteger.ONE)

&& alice.getM().gcd(n).equals(BigInteger.ONE)

&& alice.getK().gcd(n).equals(BigInteger.ONE)

&& a.gcd(alice.findCoprime(a)).equals(BigInteger.ONE));

}

}