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 ЭЦП

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 подписи DSA.

2 Теоретические сведения

DSA (англ. Digital Signature Algorithm – алгоритм цифровой подписи) – криптографический алгоритм с использованием закрытого ключа (из пары ключей: <открытый; закрытый>) для создания электронной подписи, но не для шифрования (в отличие от RSA и схемы Эль-Гамаля). Подпись создается секретно (закрытым ключом), но может быть публично проверена (открытым ключом). Это означает, что только один субъект может создать подпись сообщения, но любой может проверить её корректность. Алгоритм основан на вычислительной сложности взятия логарифмов в конечных полях.

Алгоритм DSA представляет собой вариант алгоритмов подписи Шнорра и Эль-Гамаля. В алгоритме используются следующие параметры:

- $p =$ простое число длиной L битов, где L принимает значение, кратное 64, в диапазоне от 512 до 1024. (В первоначальном стандарте размер p был фиксирован и равен 512 битам. Это ограничение вызвало множество критических замечаний, и NIST отменил его).

- $q =$ 160-битовый простой множитель $p - 1$.

- $g = h^{\frac{p-1}{q}} \bmod p$, где h - любое числа, меньшее $p - 1$, для которого $h^{\frac{p-1}{q}} \bmod p$ больше 1.

- $x =$ число, меньшее q (160-битовое число).

- $y = g^x \bmod p$.

В алгоритме также используется однонаправленная хэш-функция $H(m)$. Стандарт определяет использование алгоритма SHA-1 (первые версии стандартов).

Первые три параметра p, q, g открыты и могут быть общими для пользователей сети. Закрытым ключом является x, а открытым – y. Чтобы подписание сообщение m, Алиса пользуется следующим алгоритмом.

Алгоритм работы схемы подписи DSA:

Вход: L – длина числа p и сообщение m.

Выход: «Подпись прошла проверку» либо «Подпись не прошла проверку».

Шаг 1. Алиса генерирует случайное число $k < q$.

Шаг 2. Алиса вычисляет $r = (g^k \bmod p) \bmod q$ и $s = \left(k^{-1}(H(m) + xr)\right) \bmod q$. Ее подписью служат параметры $\{r, s\}$, которые она посылает Бобу.

Шаг 3.Боб проверяет подпись, вычисляя:

$$w = s^{-1} \bmod q$$

$$u_1 = (H(m) * w) \bmod q$$

$$u_2 = (rw) \bmod q$$

$$v = \left((g^{u_1} * y^{u_2}) \bmod p\right) \bmod q.$$

Если $v = r$, то результат «Подпись прошла проверку», иначе результат «Подпись не прошла проверку».

Доказательство корректности:

Для подписи сообщения вычисляется

$$s = \left(k^{-1}(H(m) + xr)\right) \bmod q.$$

Следовательно,

$$k = H(m) * s^{-1} + xrs^{-1} = H(m)w + xrw \bmod q.$$

Так как g имеет порядок q, то получим

$$g^k = g^{H(m)w \bmod q} g^{xrw \bmod q} = g^{H(m)w \bmod q} y^{rw \bmod q} = g^{u1} y^{u2} \bmod p.$$

Наконец, корректность схемы DSA следует из

$$r = (g^k \bmod p) \bmod q = (g^{u1} y^{u2} \bmod p) \bmod q = v.$$

3 Результаты работы

3.1 Сведения о программе

Программа была реализована на языке программирования Java. В ней есть 5 классов: DSA, $DSAService$, $Alice$, SHA и $DSAParameterGeneration$.

В классе DSA происходит считывание входных параметров: числа L и сообщения m.

Класс $DSAService$ – класс реализации самого протокола. Для инициализации передаются L и m. При инициализации генерируются параметры p, q, g, h, y, а также создается объект класса $Alice$. Каждому шагу алгоритма выше соответствует собственный метод.

Класс $Alice$ – класс участника Алиса. Для создания объекта этого класса нужно передать объект $DSAService$. При инициализации объекта данного класса генерируются закрытый ключ x и случайное число k. Также вычисляется подпись $\{r, s\}$.

Класс SHA – класс для вычисления хэш-функции по протоколу SHA-1. В классе описан следующий метод:

- `public static BigInteger encryptMessage(String message)` – вычисления хэш значения для сообщения.

Класс $DSAParameterGeneration$ – класс с описанием методов для генерации p, q, g, h. В нем описаны 2 метода:

- `public static BigInteger[] generatePAndQ(int pSizeInBits, int qSizeInBits)` – генерация простых чисел p и q.

- `static BigInteger[] findH(BigInteger p, BigInteger q)` – генерация g и h для простых чисел p и q.

3.2 Тестирование программы

Для реализации модульного тестирования были написаны тестовые классы $DSAServiceTest$, $DSAParameterGenerationTest$ и $SHATest$.

В классе $DSAServiceTest$ содержится 2 метода для тестирования самого протокола: один негативный и один положительный. В негативном тесте проверяется случай вызова исключения, когда подпись не прошла проверку. В положительном – успешная проверка подписи. Результат отработки теста представлен на рисунке 1.

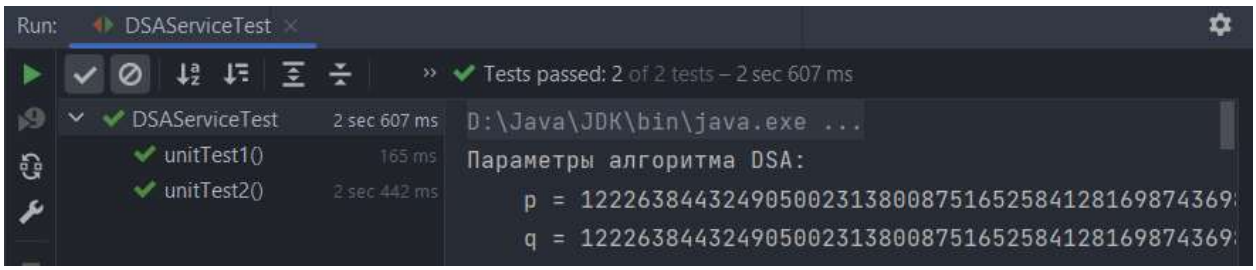

Рисунок 1 – Запуск $DSASericeTest$

В классе $DSAParameterGenerationTest$ содержится 2 тестовых метода: генерация простых чисел и их проверка по условию алгоритма; проверка генерации g и h. Результат отработки тестов представлен на рисунке 2.

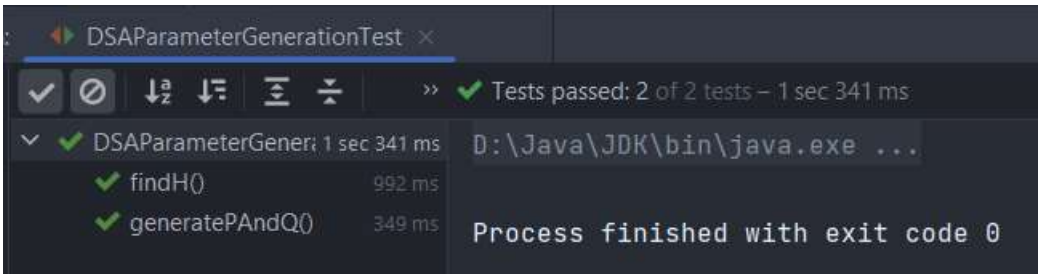

Рисунок 2 – Запуск $DSAParameterGenerationTest$

В классе $SHATest$ описан метод с проверкой получения значения хэш-функции. Проверяется, что хэш от 2х различных сообщений различен и длина составляет 160 бит. Результат отработки теста представлен на рисунке 3.

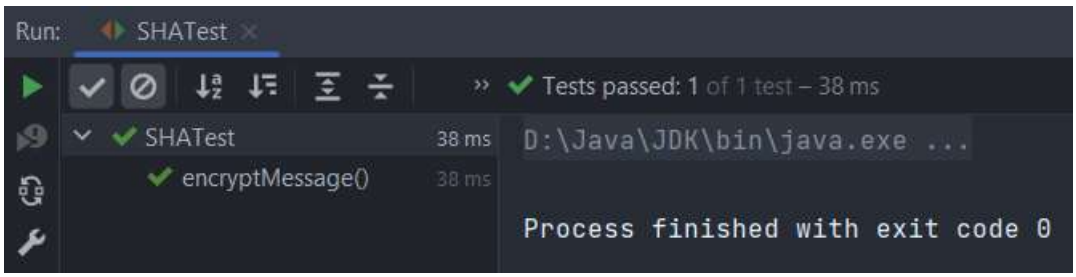

Рисунок 3 – Запуск $SHATest$

На рисунке 4 представлено негативное тестирование программы.

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 5. DSA\out\artifacts\DSA_jar> java -jar DSA.jar
Входные параметры отсутсвуют
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 5. DSA\out\artifacts\DSA_jar> java -jar DSA.jar 100
Передано некорректное число параметров.
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 5. DSA\out\artifacts\DSA_jar> java -jar DSA.jar 511 test
Длина числа p в битах должна быть в промежутке от 512 до 1024.
Ошибка в чтении входных параметров.
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 5. DSA\out\artifacts\DSA_jar> java -jar DSA.jar 1025 test
Длина числа p в битах должна быть в промежутке от 512 до 1024.
Ошибка в чтении входных параметров.
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 5. DSA\out\artifacts\DSA_jar> java -jar DSA.jar 520 test
Длина числа p в битах должна быть кратна 64
Ошибка в чтении входных параметров.
```

Рисунок 4 – Негативное тестирование

На рисунке 5 представлено положительное тестирование программы.

Рисунок 5 – Положительное тестирование (входные параметры: 1024, Test message)

ПРИЛОЖЕНИЕ А

Листинг программы

```
import java.security.SignatureException;

public class DSA {
    public static void main(String[] args) throws SignatureException {
        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- длина в битах числа p от 512 до 1024 (кратно 64)
                    \t- сообщение Алисы""");
            return;
        }
        if (args.length < 2){
            System.out.println("Передано некорректное число
параметров.");
            return;
        }
        int pSizeInBits;
        String message;
        try {
            pSizeInBits = Integer.parseInt(args[0]);
            if (pSizeInBits < 512 || pSizeInBits > 1024){
                System.out.println("Длина числа p в битах должна быть в
промежутке от 512 до 1024.");
                throw new IllegalArgumentException();
            }
            if (pSizeInBits % 64 != 0){
                System.out.println("Длина числа p в битах должна быть
кратна 64");
                throw new IllegalArgumentException();
            }
            message = args[1];
        } catch (IllegalArgumentException e) {
            System.out.println("Ошибка в чтении входных параметров.");
            return;
        }
        DSAService service = new DSAService(pSizeInBits, message);
    }
}


import java.math.BigInteger;
import java.security.SignatureException;

public class DSAService {
    private final BigInteger p, q, g, h, y;
    private BigInteger v;
    private final String message;
    Alice alice;
```

```
    public DSAService(int pSizeInBits, String message) throws
SignatureException {
        this.message = message;
        BigInteger[] pq =
DSAParameterGeneration.generatePAndQ(pSizeInBits, 160);
        this.p = pq[0];
        this.q = pq[1];
        BigInteger[] gh = DSAParameterGeneration.findH(p, q);
        this.g = gh[0];
        this.h = gh[1];
        alice = new Alice(this);
        this.y = g.modPow(alice.getX(), p);
        System.out.println(this);
        this.step1();
    }

    @Override
    public String toString() {
        return String.format("""
                Параметры алгоритма DSA:
                \tp = %d
                \tq = %d
                \th = %d
                \tg = %d
                \tx = %d
                \ty = %d\n""", p, q, h, g, alice.getX(), y);
    }

    private void step1() throws SignatureException {
        System.out.printf("""
                \n--- Шаг 1 ---
                Алиса хочет подписать следующее сообщение: %s
                Она генерирует число k = %d\n""", message,
alice.getK());
        this.step2();
    }

    private void step2() throws SignatureException {
        System.out.printf("""
                \n--- Шаг 2 ---
                Алиса генерирует подпись:
                   r = %d
                   s = %d
                Затем посылает ее Бобу\n""", alice.getR(),
alice.getS());
        this.step3(alice.getR(), alice.getS());
    }

    private void step3(BigInteger r, BigInteger s) throws
SignatureException {
        BigInteger w = s.modInverse(q);
        BigInteger u1 =
(SHA.encryptMessage(message).multiply(w)).mod(q);
        BigInteger u2 = r.multiply(w).mod(q);
        v = g.modPow(u1, p).multiply(y.modPow(u2, p)).mod(p).mod(q);
        System.out.printf("""
                \n--- Шаг 3 ---
                Боб проверяет подпись, вычисляя:
```

```
                    w = %d
                    u1 = %d
                    u2 = %d
                    v = %d\n""", w, u1, u2, v);
        if (!v.equals(r)) {
            throw new SignatureException("Подпись не прошла проверку!\nv
= " + v + "\nr = " + r);
        }
        System.out.println("Подпись прошла проверку.");
    }


    public BigInteger getP() {
        return p;
    }

    public BigInteger getQ() {
        return q;
    }

    public BigInteger getG() {
        return g;
    }

    public String getMessage() {
        return message;
    }

    public Alice getAlice() {
        return alice;
    }

    public BigInteger getV() {
        return v;
    }
}


import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import java.security.SignatureException;

import static org.junit.jupiter.api.Assertions.assertThrows;

class DSAServiceTest {
    @Test
    public void unitTest1() {
        java.security.SignatureException thrown = assertThrows(
                java.security.SignatureException.class,
                () -> new DSAService(100, "test"),
                "Ожидалось исключение в new DSAService(), но его не
было."
        );
    }

    @Test
    public void unitTest2() throws SignatureException {
```

```
        DSAService service = new DSAService(1024, "test");
        Alice alice = service.getAlice();
        Assertions.assertEquals(alice.getR(), service.getV());
    }

}
```

```
import java.math.BigInteger;
import java.security.SecureRandom;

public class Alice {
    private final BigInteger k, r, s;
    private BigInteger x;

    public Alice(DSAService dsa) {
        SecureRandom rnd = new SecureRandom();
        this.x      =      new      BigInteger(dsa.getQ().bitLength(),
rnd).mod(dsa.getQ());
        this.x     =     x.multiply(BigInteger.ONE.shiftLeft(160     -
x.bitLength()));
        this.k      =      new      BigInteger(dsa.getQ().bitLength(),
rnd).mod(dsa.getQ());
        this.r = dsa.getG().modPow(k, dsa.getP()).mod(dsa.getQ());
        this.s = (k.modInverse(dsa.getQ())

.multiply(SHA.encryptMessage(dsa.getMessage()).add(x.multiply(r))))
                .mod(dsa.getQ());
    }

    public BigInteger getX() {
        return x;
    }

    public BigInteger getK() {
        return k;
    }

    public BigInteger getR() {
        return r;
    }

    public BigInteger getS() {
        return s;
    }
}
```

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA {
    public static BigInteger encryptMessage(String message) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            byte[] messageDigest = md.digest(message.getBytes());
            return new BigInteger(1, messageDigest);
```

```
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
}


import org.junit.jupiter.api.Test;

import java.math.BigInteger;

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

class SHATest {

    @Test
    void encryptMessage() {
        BigInteger hash1 = SHA.encryptMessage("test");
        BigInteger hash2 = SHA.encryptMessage("test test");
        assertTrue(hash1.bitLength() == 160 && hash2.bitLength() == 160
&& !hash1.equals(hash2));
    }
}


import java.math.BigInteger;
import java.security.SecureRandom;


public class DSAParameterGeneration {
    public static BigInteger[] generatePAndQ(int pSizeInBits, int
qSizeInBits) {
        SecureRandom random = new SecureRandom();
        BigInteger q = BigInteger.probablePrime(qSizeInBits, random);
        BigInteger k = BigInteger.ONE;
        int diff = pSizeInBits - qSizeInBits;
        if (diff > 0) {
            k = BigInteger.ONE.shiftLeft(pSizeInBits - qSizeInBits);
        }
        BigInteger p = q.multiply(k).add(BigInteger.ONE);
        if (diff > 0) {
            while (!p.isProbablePrime(100)) {
                q = BigInteger.probablePrime(qSizeInBits, random);
                p = q.multiply(k).add(BigInteger.ONE);
            }
        }
        return new BigInteger[]{p, q};
    }

    static BigInteger[] findH(BigInteger p, BigInteger q) {
        SecureRandom rnd = new SecureRandom();
        BigInteger h = new BigInteger("2");
        BigInteger g = h.modPow((p.subtract(BigInteger.ONE)).divide(q),
p);
        while (g.compareTo(BigInteger.ONE) <= 0) {
            h = h.add(BigInteger.ONE).mod(p);
            g = h.modPow((p.subtract(BigInteger.ONE)).divide(q), p);
        }
```

```
        return new BigInteger[]{g, h};
    }
}


import org.junit.jupiter.api.Test;

import java.math.BigInteger;

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

class DSAParameterGenerationTest {

    @Test
    void generatePAndQ() {
        BigInteger[]  pq  =  DSAParameterGeneration.generatePAndQ(512,
160);
        assertTrue(pq.length == 2
                && pq[0].isProbablePrime(50)
                && pq[1].isProbablePrime(50)
                &&
(pq[0].subtract(BigInteger.ONE)).mod(pq[1]).equals(BigInteger.ZERO));
    }

    @Test
    void findH() {
        BigInteger[]  pq  =  DSAParameterGeneration.generatePAndQ(512,
160);
        BigInteger[] gh = DSAParameterGeneration.findH(pq[0], pq[1]);
        assertTrue(gh.length == 2 && gh[0].compareTo(BigInteger.ONE) >
0);
    }
}
```