Министерство образования Республики Беларусь
Белорусский государственный университет информатики и радиоэлектроники
Кафедра информатики

ОТЧЕТ
по лабораторной работе №6
Цифровая подпись

Выполнил:
студент гр. 653501
Шинкевич Г. С.

Проверил:
Артемьев В. С.

Минск 2019

ЗАДАНИЕ:

Реализовать программное средство формирования и проверки ЭЦП на базе алгоритма ГОСТ 3410.

ГОСТ 34.10-2018

Действующий межгосударственный криптографический стандарт, описывающий алгоритмы формирования и проверки электронной цифровой подписи реализуемой с использованием операций в группе точек эллиптической кривой, определенной над конечным простым полем.

Цифровая подпись позволяет:
1. Аутентифицировать лицо, подписавшее сообщение
2. Контролировать целостность сообщения
3. Защищать сообщение от подделок

Все стандарты семейства 34.10 основаны на эллиптических кривых. Стойкость этих алгоритмов основывается на сложности вычисления дискретного логарифма в группе точек эллиптической кривой, а также на стойкости соотвествующей хеш-функции.

После подписывания сообщения M к нему дописывается цифровая подпись размером 512 или 1024 бит и текстовое поле. В текстовом поле могут содержаться, например, дата и время отправки или различные данные об отправителе.

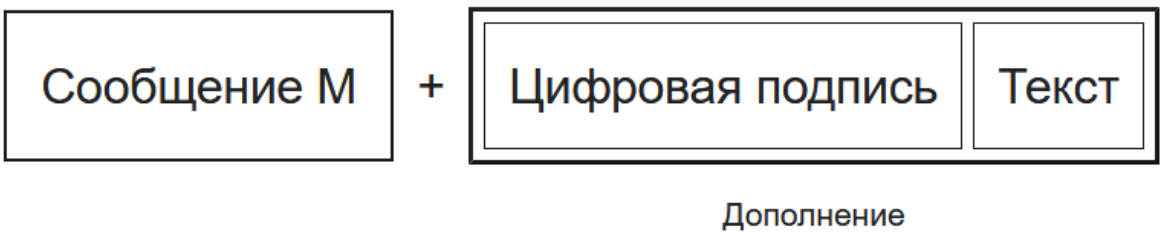

Рисунок 1. Добавление цифровой подписи

Параметры алгоритма

- простое число p — модуль эллиптической кривой
- эллиптическая кривая E задаётся коэффициентами a, b ∈ Fp, где Fp — конечное поле из p элементов
- простое число q, порядок некоторой циклической подгруппы группы точек эллиптической кривой

- точка $P = (xP , yP)$ эллиптической кривой E, являющаяся генератором подгруппы порядка q, то есть $q \cdot P = 0$ и $k \cdot P \neq 0$ для всех $k = 1, 2, \ldots, q-1$, где 0 — нейтральный элемент группы точек эллиптической кривой E.
- h(M) — хеш-функция, которая отображает сообщения M в двоичные векторы длины 256 бит.
- ключ шифрования d — целое число, лежащее в пределах $0<d<q$
- ключ расшифрования $Q = (xQ, yQ)$, вычисляемый как $Q = d \cdot P$

Алгоритм формирования цифровой подписи

1. Вычисление хеш-функции от сообщения M: $h = h(M)$.

2. Вычисление $e = z \bmod q$, и если $e = 0$, положить $e = 1$. Где z — целое число, соответствующее h.

3. Генерация случайного числа k такого, что $0 < k < q$.

4. Вычисление точки эллиптической кривой $C = kP$, и по ней нахождение $r = x_c \bmod q$, где x_c — это координата x точки C. Если $r=0$, возвращаемся к предыдущему шагу.

5. Нахождение $s = (rd + ke) \bmod q$. Если $s=0$, возвращаемся к шагу 3.

6. Формирование цифровой подписи $\xi = (r \mid s)$.

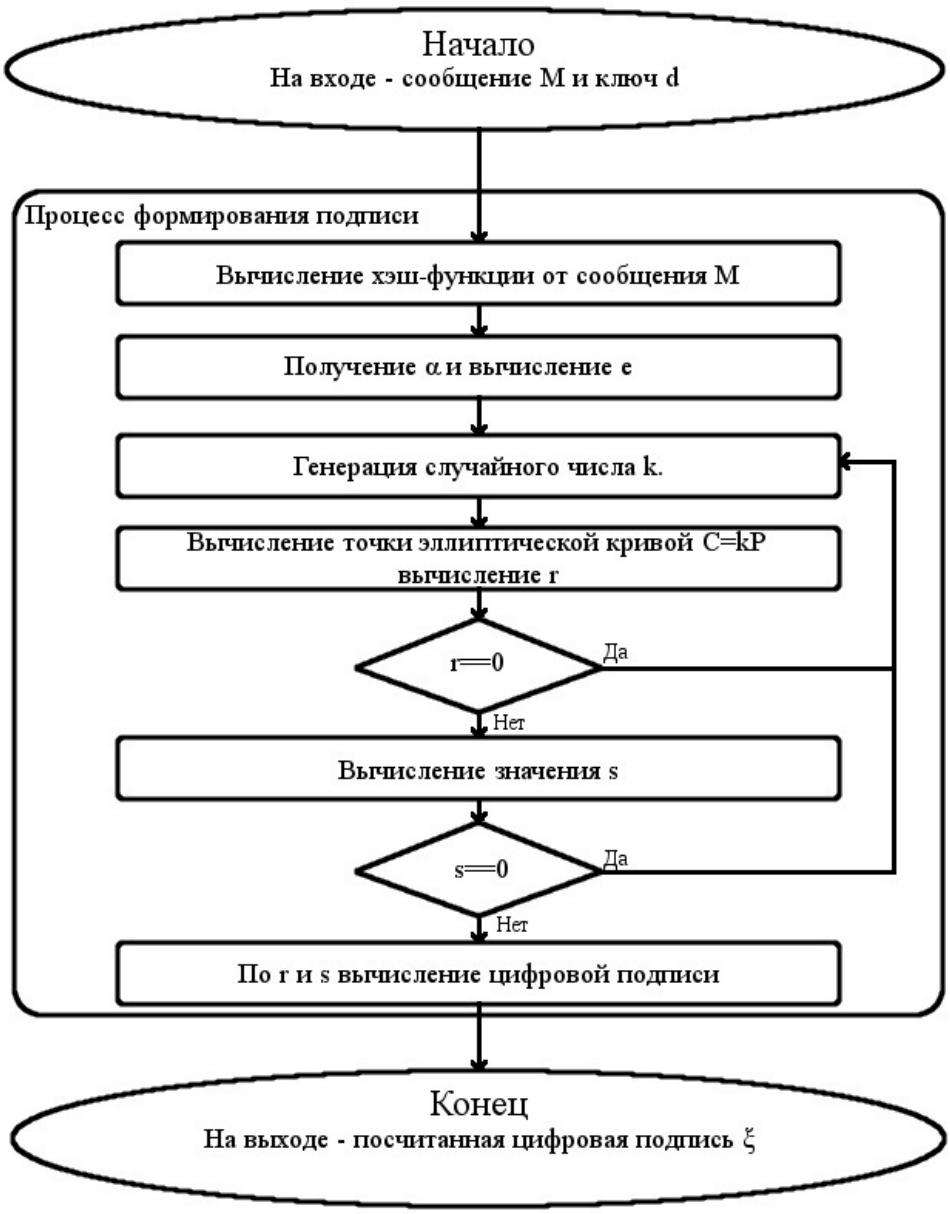

Рисунок 2. Алгоритм формирования цифровой подписи

Алгоритм проверки цифровой подписи

1. Вычисление по цифровой подписи ξ чисел r и s. Если хотя бы одно из неравенств r<q и s<q неверно, то подпись неправильная.

2. Вычисление хеш-функции от сообщения M: h = h (M).

3. Вычисление e = z mod q, и если e = 0, положить e = 1. Где z — целое число, соответствующее h.

4. Вычисление $\nu = e^{-1} \bmod q$.

5. Вычисление z1 = sv mod q и z2 = − rv mod q.

6. Вычисление точки эллиптической кривой C = z1 * P + z2 * Q. И определение R = x_c mod q, где x_c — координата x точки C.

7. В случае равенства R=r подпись правильная, иначе — неправильная.

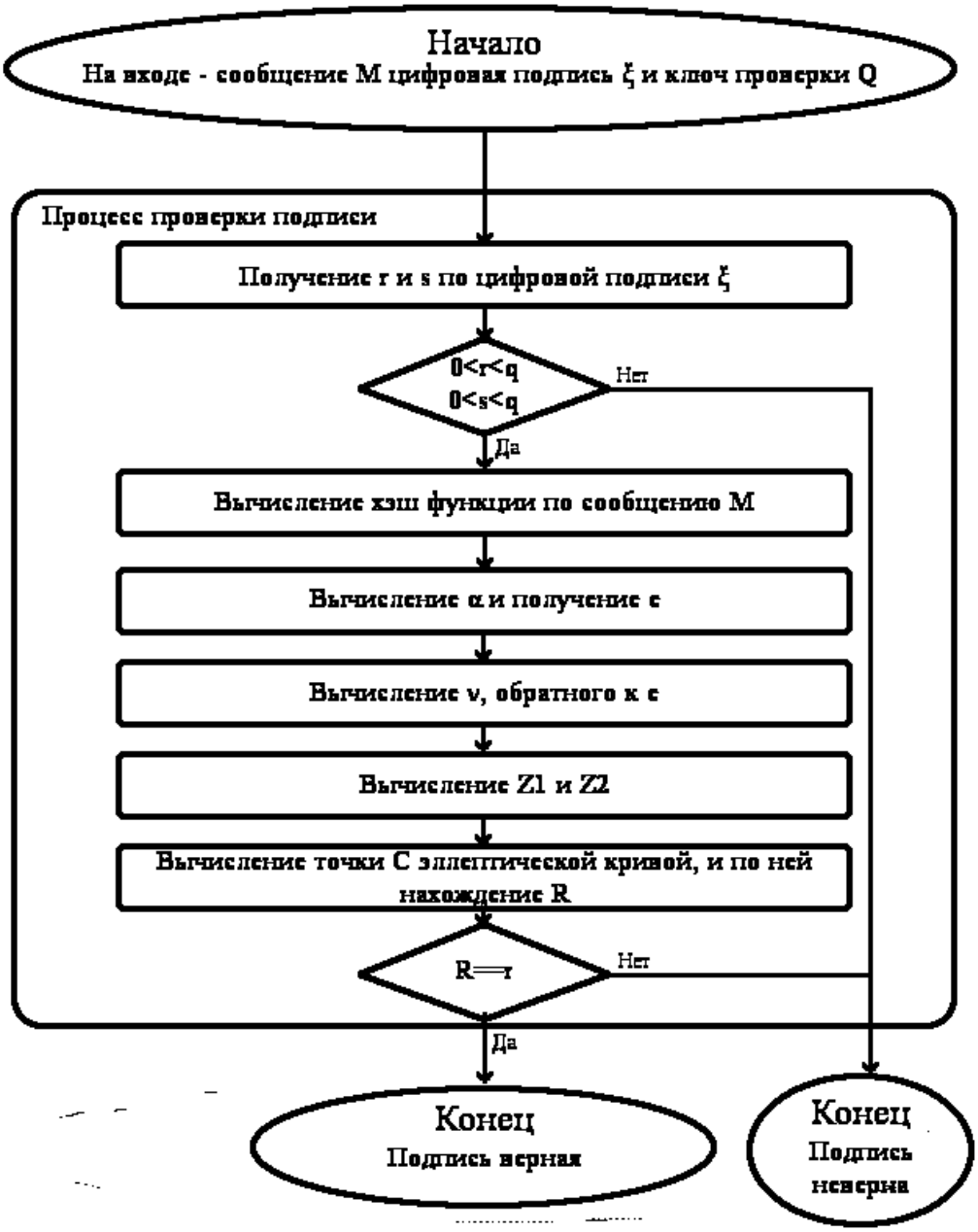

Рисунок 3. Алгоритм проверки цифровой подписи

ДЕМОНСТРАЦИЯ РАБОТЫ

```
Test signature: (41aa28d2f1ab148280cd9ed56feda41974053554a42767b83ad043fd39dc0493,
  1456c64ba4642a1653c235a98a60249bcd6d3f746b631df928014f6c5bf9c40)
Correct test signature

Message: BSUIR
Signature: (2c0e17a6f0c7bad126456ba644bf4c1ef2770a61aea7636fa29280e396411ba3,
  55b333a668a45443f74bc41a127f72b60397b9e3887c22130938250157f09c4)
Correct signature
```

ВЫВОД

В результате лабораторной работы была написана программа для формирования и проверки ЭЦП на базе алгоритма ГОСТ 3410.

ЛИСТИНГ ПРОГРАММЫ

```
import hashlib
import random


class Point:
    def __init__(self, a, b, p, q, x, y):
        self.a = a
        self.b = b
        self.p = p
        self.q = q
        self.x = x
        self.y = y

    def __add__(self, other):
        if (self.x, self.y) == (0, 0):
            return other
        if (other.x, other.y) == (0, 0):
            return self
        if self.x == other.x and (self.y != other.y or self.y == 0):
            return Point(self.a, self.b, self.p, self.q, 0, 0)
        if self.x == other.x and self.y == other.y:
            m = (3 * self.x ** 2 + self.a) * multiplicative_inverse(2 * self.y,
self.p)
        else:
            m = (self.y - other.y) * multiplicative_inverse(self.x - other.x,
self.p)

        x_r = (m ** 2 - self.x - other.x) % self.p
        y_r = (m * (self.x - x_r) - self.y) % self.p

        return Point(self.a, self.b, self.p, self.q, x_r, y_r)

    def __mul__(self, number):
        result = Point(self.a, self.b, self.p, self.q, 0, 0)
        addend = self

        for bit in bits(number):
```

```
            if bit == 1:
                result += addend
            addend += addend
        return result


def bits(number):
    while number:
        yield number & 1
        number >>= 1


def get_bezout_coeffs(a, b):
    # ax + by = gcd(a, b)
    x, x_, y, y_ = 1, 0, 0, 1
    while b:
        q = a // b
        a, b = b, a % b
        x, x_ = x_, x - x_ * q
        y, y_ = y_, y - y_ * q
    return x, y


def multiplicative_inverse(a, b):
    x, y = get_bezout_coeffs(a, b)
    if x < 0:
        return b + x
    return x


def sign(d, message, q, P):
    h = hashlib.sha256(message).digest()

    z = int.from_bytes(h, byteorder='big')
    e = z % q
    if e == 0:
        e = 1

    s = 0
```

```
    while True:
        k = random.randint(1, q - 1)
        C = P * k
        r = C.x % q
        if r == 0:
            continue
        s = (r * d + k * e) % q
        if s != 0:
            break

    return r, s


def verify(Q, message, signature, q, P):
    r, s = signature
    if not r < q or not s < q:
        return False

    h = hashlib.sha256(message).digest()

    z = int.from_bytes(h, byteorder='big')
    e = z % q
    if e == 0:
        e = 1

    v = multiplicative_inverse(e, q)

    z1 = s * v % q
    z2 = -r * v % q

    C = P * z1 + Q * z2
    R = C.x % q

    if R == r:
        return True
    else:
        return False
```

```
def sign_test(d, q, P, e, k):
    s = 0
    while True:
        C = P * k
        r = C.x % q
        if r == 0:
            continue
        s = (r * d + k * e) % q
        if s != 0:
            break

    return r, s


def verify_test(Q, signature, q, P, e):
    r, s = signature
    if not r < q or not s < q:
        return False

    v = multiplicative_inverse(e, q)

    z1 = s * v % q
    z2 = -r * v % q

    C = P * z1 + Q * z2
    R = C.x % q

    if R == r:
        return True
    else:
        return False


if __name__ == '__main__':
    p =
0x8000000000000000000000000000000000000000000000000000000000000431
    a = 0x7
```

```
    b =
0x5FBFF498AA938CE739B8E022FBAFEF40563F6E6A3472FC2A51
4C0CE9DAE23B7E
    q =
0x800000000000000000000000000000150FE8A1892976154C59CFC
193ACCF5B3
    x = 0x2
    y =
0x8E2A8A0E65147D4BD6316030E16D19C85C97F0A9CA267122B9
6ABBCEA7E8FC8
    d =
0x7A929ADE789BB9BE10ED359DD39A72C11B60961F49397EEE1
D19CE9891EC3B28
    P = Point(a, b, p, q, x, y)
    Q = P * d

    e =
0x2DFBC1B372D89A1188C09C52E0EEC61FCE52032AB1022E8E67
ECE6672B043EE5
    k =
0x77105C9B20BCD3122823C8CF6FCC7B956DE33814E95B7FE64F
ED924594DCEAB3
    test_signature = sign_test(d, q, P, e, k)
    test_is_correct = verify_test(Q, test_signature, q, P, e)
    print(f"Test signature: ({test_signature[0]:x}, {test_signature[1]:x})")
    if test_is_correct:
        print("Correct test signature")
    else:
        print("Wrong test signature")
    print()
    message = "BSUIR"
    signature = sign(d, message.encode(), q, P)
    is_correct = verify(Q, message.encode(), signature, q, P)
    print(f"Message: {message}")
    print(f"Signature: ({signature[0]:x}, {signature[1]:x})")
    if is_correct:
        print("Correct signature")
    else:
        print("Wrong signature")
```