

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ОТЧЕТ  
по лабораторной работе  
на тему  
**Цифровая подпись**

Выполнил студент группы 053501  
Криштафович Карина Дмитриевна

---

(подпись)

Проверил  
ассистент кафедры информатики  
Лещенко Евгений Александрович

---

(подпись)

Минск 2023

## СОДЕРЖАНИЕ

Введение.....	3
1 Демонстрация работы программы.....	4
2 Теоретические сведения.....	5
Заключение.....	9
Приложение А (обязательное) Листинг программного кода.....	10

## **ВВЕДЕНИЕ**

Цифровая подпись - это один из ключевых элементов современной криптографии, который обеспечивает аутентификацию и целостность данных в цифровой среде. Она играет важную роль в обеспечении безопасности электронных коммуникаций, электронной коммерции, систем передачи данных и многих других областях. В рамках данной лабораторной работы мы будем исследовать и изучать один из криптографических стандартов, широко применяемых в России - ГОСТ 34.10.

ГОСТ 34.10 является российским стандартом для цифровой подписи, разработанным с целью обеспечения безопасной передачи и хранения данных, а также аутентификации пользователей и защиты информации. Он определяет алгоритмы и процедуры для создания и проверки электронных подписей, которые обеспечивают высокую стойкость к атакам и обеспечивают доверие в цифровом мире.

В этой лабораторной работе мы рассмотрим основные принципы работы ГОСТ 34.10, изучим его математические основы, исследуем процесс создания и верификации цифровых подписей с использованием этого стандарта. Мы также рассмотрим практические аспекты его применения и роль, которую он играет в обеспечении информационной безопасности в России и за её пределами.

# 1 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

```
p = 57896044618658097711785492504343953926634992332820282019728792003956564821041
a = 7
b = 43308876546767276905765904595650931995942111794451039583252968842033849580414
q = 57896044618658097711785492504343953927082934583725450622380973592137631069619
x = 2
y = 4018974056539037503335449422937059775635739389905545080690979365213431566280
```

Рисунок 1 – Начальные данные

Результат работы программы представлен на рисунке 2.

```
e = 45586903794418611460986632094363394605395426255537673057642288579829570008197
r = 21626567202189741737690437610133319091352376597687155615312619115852251807853
q = 57896044618658097711785492504343953927082934583725450622380973592137631069619
еср: 21626567202189741737690437610133319091352376597687155615312619115852251807853
ЭЦП подтверждена
```

Рисунок 2 – Результат работы программы

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Блок-схема алгоритма электронно-цифровой подписи представлена на рисунке 3.



Рисунок 3 – алгоритм электронно-цифровой подписи

Блок-схема алгоритма формирования цифровой подписи представлена на рисунке 4.

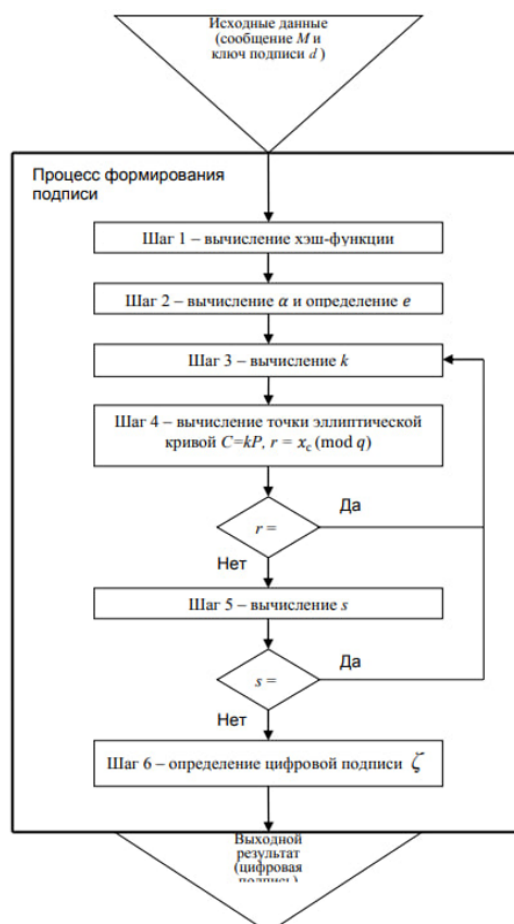


Рисунок 4 – алгоритм формирования цифровой подписи

Блок-схема алгоритма верификации цифровой подписи представлена на рисунке 5.

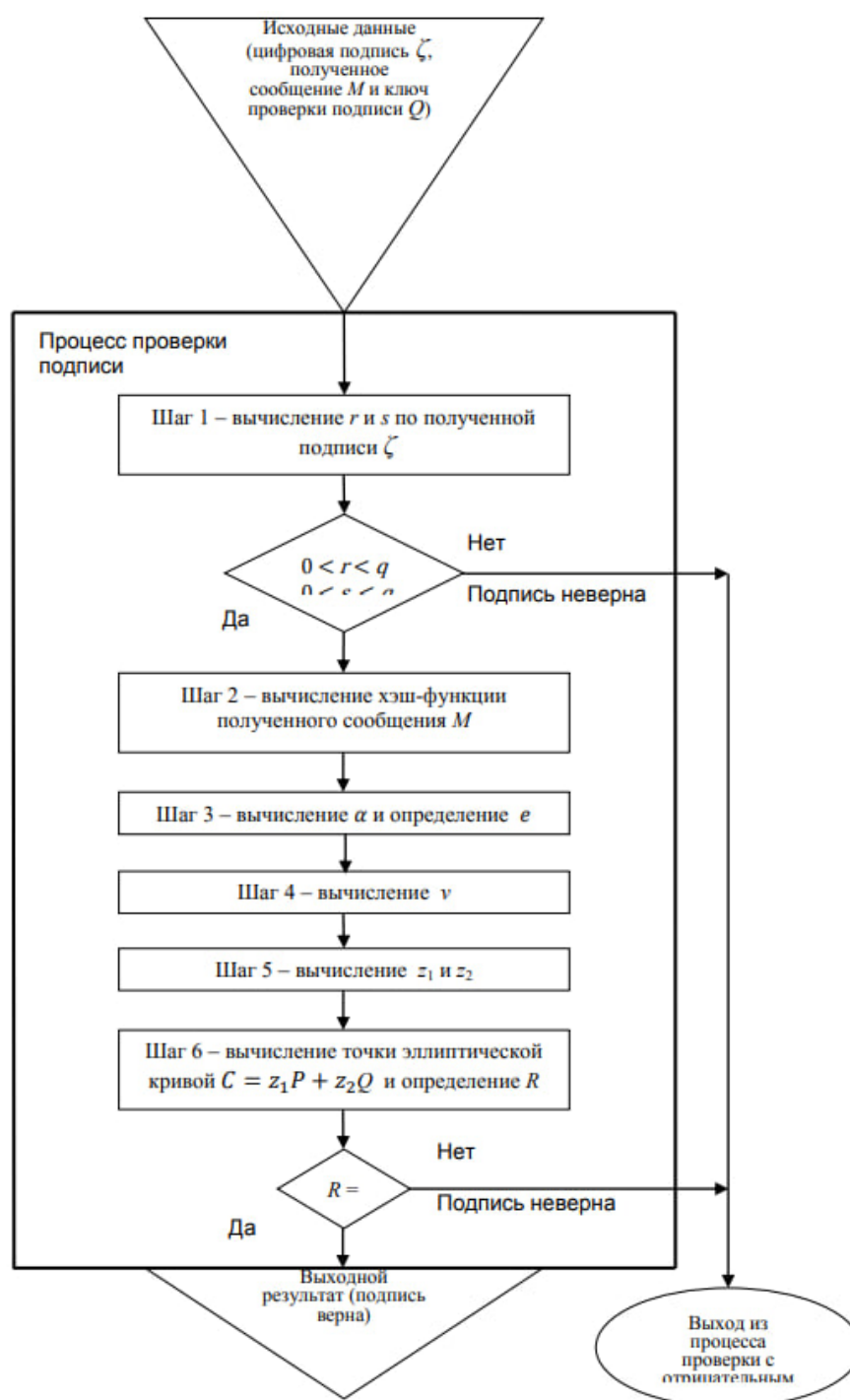


Рисунок 5 – алгоритм верификации цифровой подписи

Для получения цифровой подписи под сообщением  $M \in V^*$  необходимо выполнить следующие действия (шаги) по алгоритму I:

Шаг 1 – вычислить хэш-код сообщения  $M : \bar{h} = h(M)$ . (14)

Шаг 2 – вычислить целое число  $\alpha$ , двоичным представлением которого является вектор  $\bar{h}$ , и определить

$$e \equiv \alpha \pmod{q}. \quad (15)$$

Если  $e = 0$ , то определить  $e = 1$ .

Шаг 3 – сгенерировать случайное (псевдослучайное) целое число  $k$ , удовлетворяющее неравенству

$$0 < k < q. \quad (16)$$

Шаг 4 – вычислить точку эллиптической кривой  $C = kP$  и определить

$$r \equiv x_c \pmod{q}, \quad (17)$$

где  $x_c$  –  $x$ -координата точки  $C$ .

Если  $r = 0$ , то вернуться к шагу 3.

Шаг 5 – вычислить значение

$$s \equiv (rd + ke) \pmod{q}. \quad (18)$$

Если  $s = 0$ , то вернуться к шагу 3.

Шаг 6 – вычислить двоичные векторы  $\bar{r}$  и  $\bar{s}$ , соответствующие  $r$  и  $s$ , и определить цифровую подпись  $\zeta = (\bar{r} \parallel \bar{s})$  как конкатенацию двух двоичных векторов.

Исходными данными этого процесса являются ключ подписи  $d$  и подписываемое сообщение  $M$ , а выходным результатом – цифровая подпись  $\zeta$ .

Для проверки цифровой подписи  $\zeta$  под полученным сообщением  $M$  необходимо выполнить следующие действия (шаги) по алгоритму II:

Шаг 1 – по полученной подписи  $\zeta$  вычислить целые числа  $r$  и  $s$ . Если выполнены неравенства  $0 < r < q$ ,  $0 < s < q$ , то перейти к следующему шагу. В противном случае подпись неверна.

Шаг 2 – вычислить хэш-код полученного сообщения  $M$

$$\bar{h} = h(M). \quad (19)$$

Шаг 3 – вычислить целое число  $\alpha$ , двоичным представлением которого является вектор  $\bar{h}$  и определить

$$e \equiv \alpha \pmod{q}. \quad (20)$$

Если  $e = 0$ , то определить  $e = 1$ .

Шаг 4 – вычислить значение  $v \equiv e^{-1} \pmod{q}$ . (21)

Шаг 5 – вычислить значения

$$z_1 \equiv sv \pmod{q}, \quad z_2 \equiv -rv \pmod{q}. \quad (22)$$

Шаг 6 – вычислить точку эллиптической кривой  $C = z_1P + z_2Q$  и определить

$$R \equiv x_c \pmod{q}, \quad (23)$$

где  $x_c$  –  $x$ -координата точки  $C$ .

Шаг 7 – если выполнено равенство  $R = r$ , то подпись принимается, в противном случае – подпись неверна.

Исходными данными этого процесса являются подписанное сообщение  $M$ , цифровая подпись  $\zeta$  и ключ проверки подписи  $Q$ , а выходным результатом – свидетельство о достоверности или ошибочности данной подписи.

Схема процесса проверки цифровой подписи приведена на рисунке 3.



## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы по формированию и проверке ЭЦП (Электронной Цифровой Подписи) на базе алгоритма ГОСТ 34.10 были достигнуты следующие важные результаты и усвоены основные концепции криптографии:

Понимание цифровой подписи: Мы изучили суть цифровой подписи и её значение в современной криптографии. Убедились в том, что она является неотъемлемой частью обеспечения безопасности данных в цифровой эпохе, обеспечивая аутентификацию и целостность информации. Основы ГОСТ 34.10: Изучение алгоритма ГОСТ 34.10 позволило нам понять, как работает этот стандарт, включая основы асимметричной криптографии и эллиптических кривых, используемых в данном алгоритме. Мы создали программное средство для формирования и проверки ЭЦП с использованием алгоритма ГОСТ 34.10. В ходе разработки, мы изучили математические операции, необходимые для создания подписи и её проверки.

Лабораторная работа позволила нам не только приобрести теоретические знания о цифровой подписи и алгоритме ГОСТ 34.10, но и научиться создавать и проверять подписи на практике. Эти навыки и знания останутся полезными в будущих задачах обеспечения информационной безопасности и защиты данных в цифровой среде.

**ПРИЛОЖЕНИЕ А**  
**(обязательное)**  
**Листинг программного кода**

```
import random
from ec import ECPoint

class DSgost:
    def __init__(self, p, a, b, q, p_x, p_y):
        self.p_point = ECPoint(p_x, p_y, a, b, p)
        self.q = q
        self.a = a
        self.b = b
        self.p = p

    def gen_keys(self):
        d = 55441196065363246126355624130324183196576709222340016572108097750006097525544
        q_point = d * self.p_point
        return d, q_point

    def encrypt(self, message, private_key):
        e = message % self.q
        print(f"e = {e}")
        k = random.randint(1, self.q - 1)
        r, s = 0, 0
        while r == 0 or s == 0:
            c_point = k * self.p_point
            r = c_point.x % self.q
            s = (r * private_key + k * e) % self.q
        concatenated_rs = str(r) + str(s)
        return r, s, concatenated_rs

    def verify(self, message, encrypt, public_key):
        if not (0 < encrypt[0] < self.q) or not (0 < encrypt[0] < self.q):
            return False
        e = message % self.q
        if e == 0:
            e = 1
        nu = ECPoint._mod_inverse(e, self.q)
        z1 = (encrypt[1] * nu) % self.q
        z2 = (-encrypt[0] * nu) % self.q
        c_point = z1 * self.p_point + z2 * public_key
        r = c_point.x % self.q
        if r == encrypt[0]:
            return True
```

```

        return False
class ECPoint:
    def __init__(self, x=0, y=0, a=0, b=0, p=0,
is_polynomial_basis=False):
        self.x = x
        self.y = y
        self.a = a
        self.b = b
        self.p = p
        self.pol_basis = is_polynomial_basis

    # inverse int b modulo p
    @staticmethod
    def _mod_inverse(b, p):
        x0, x1, y0, y1, n = 1, 0, 0, 1, p
        while n != 0:
            q, b, n = b // n, n, b % n
            x0, x1 = x1, x0 - q * x1
            y0, y1 = y1, y0 - q * y1
        return x0 % p

    def __add__(self, other):
        p_result = ECPoint()
        p_result.a = self.a
        p_result.b = self.b
        p_result.p = self.p

        dx = (other.x - self.x) % self.p
        dy = (other.y - self.y) % self.p
        if self.x == other.x and self.y == other.y:
            l = ((3 * self.x ** 2 + self.a) * ECPoint._mod_inverse(2
* self.y, self.p)) % self.p
        else:
            if self.x == other.x:
                return float('inf')
            dx_inverse = ECPoint._mod_inverse(dx, self.p)
            l = (dy * dx_inverse) % self.p
        p_result.x = (l * l - self.x - other.x) % self.p
        p_result.y = (l * (self.x - p_result.x) - self.y) % self.p

        return p_result

    def __rmul__(self, other):
        p_result = ECPoint(self.x, self.y, self.a, self.b, self.p,
self.pol_basis)
        temp = ECPoint(self.x, self.y, self.a, self.b, self.p,
self.pol_basis)
        x = other - 1
        while x != 0:

```

```
    if x % 2 != 0:
        p_result += temp
        x -= 1
    x //= 2
    temp = temp + temp
return p_result
```