Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №6

“Цифровая подпись”

|  |  |
| --- | --- |
|  | Выполнил:  студент группы 053504  Горожанкин В.О. |
|  | Проверил  ассистент кафедры информатики  Лещенко Евгений Александрович |

Минск 2023

# СОДЕРЖАНИЕ

Введение3

[1](#_heading=h.dagojaxz0wjl) Демонстрация работы программы4

[2](#_heading=h.30j0zll) Теоретические сведения5

Заключение9

Приложение [А](#_heading=h.fcdcaca7b490) [(обязательное)](#_heading=h.xcqdfzkvnwta) [Листинг программного кода](#_heading=h.drh0t9idhpqa)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

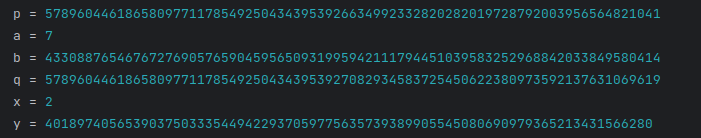


Рисунок 1 – Начальные данные

Результат работы программы представлен на рисунке 2.

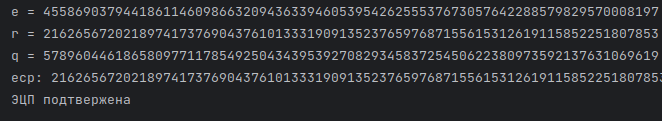
****

Рисунок 2 – Результат работы программы

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

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

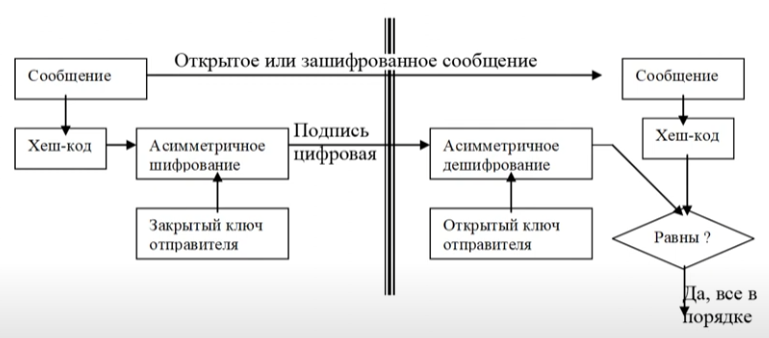


Рисунок 3 – алгоритм электронно-цифровой подписи

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

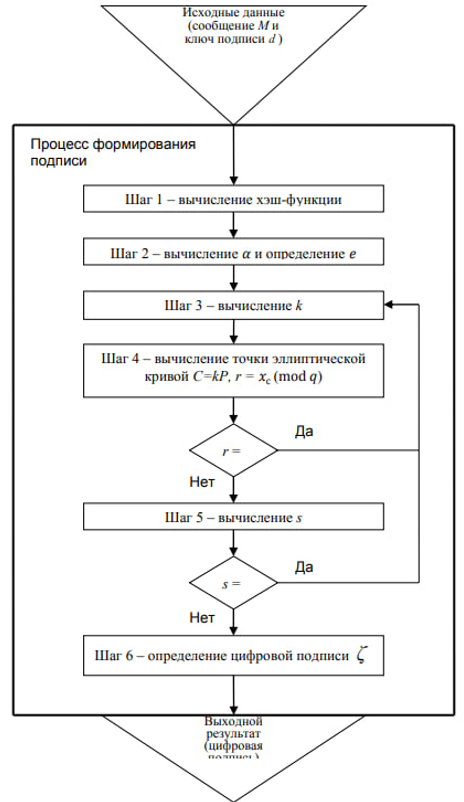


Рисунок 4 – алгоритм формирования цифровой подписи

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

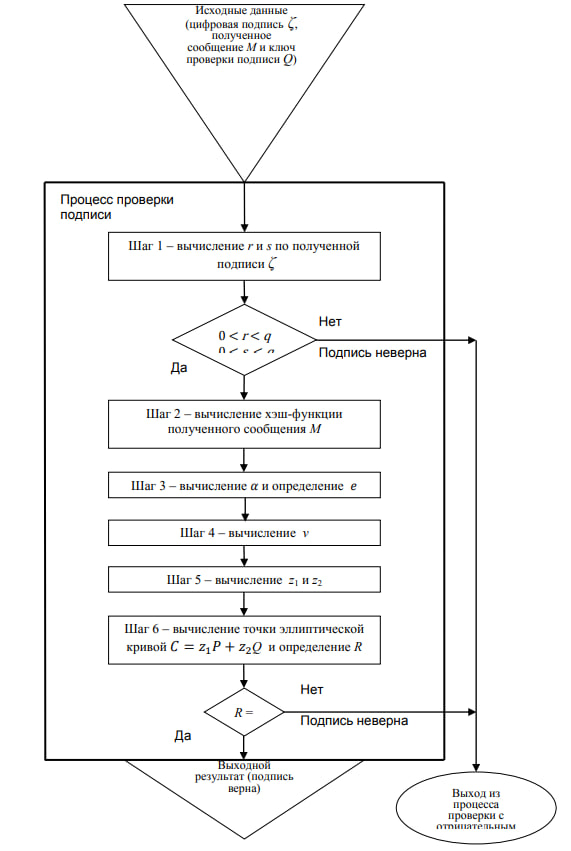
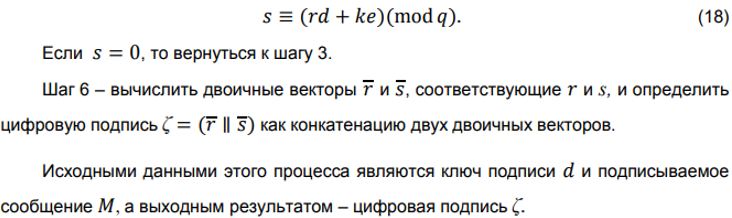
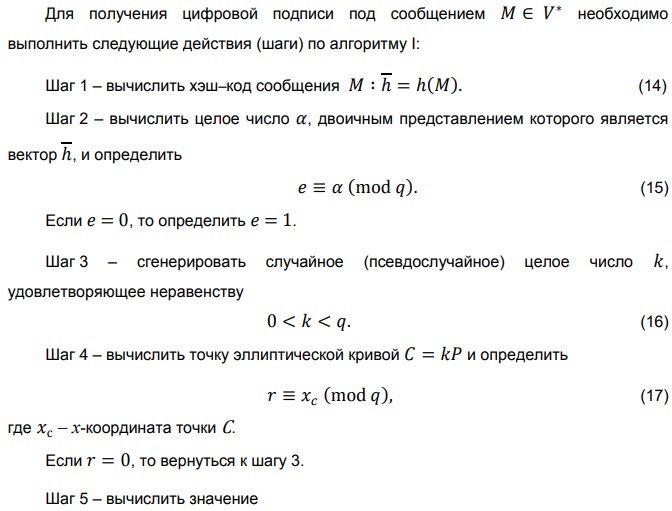
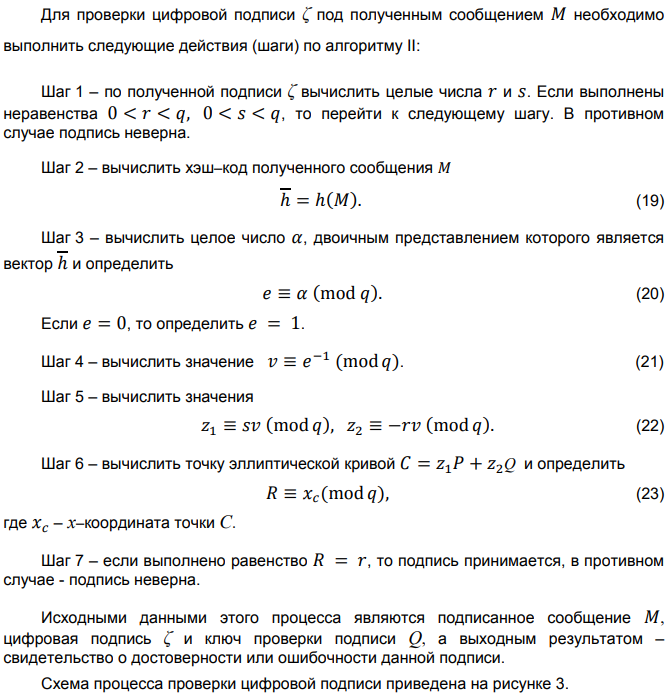


Рисунок 5 – алгоритм верификации цифровой подписи





# 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