Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №6

По дисциплине «Методы защиты информации»

По теме «Цифровая подпись»

Выполнила:

Студентка гр. 653501

Д. А. Гуринович

Проверил:

В. С. Артемьев

Минск 2019

Содержание

[1. Постановка задачи](#_2et92p0)

2. Краткие теоретические сведения

3. Схема алгоритма

4[. Результаты выполнения](#_tyjcwt)

5[. Вывод](#_3dy6vkm)

[ПРИЛОЖЕНИЕ 1. Исходный код программы.](#_1t3h5sf)

# 1. Постановка задачи

В современном мире остро стоит вопрос о безопасности. Обеспечение безопасности является важным аспектом деятельности любой компании. Для обеспечения безопасности используется множество различных средств, как аппаратных, так и программных. Одним из таких средств является подписание данных с помощью цифровой подписи.

Электронная подпись (ЭП), Электронная цифровая подпись (ЭЦП), Цифровая подпись (ЦП) позволяет подтвердить авторство электронного документа (будь то реальное лицо или, например, аккаунт в [криптовалютной](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B2%D0%B0%D0%BB%D1%8E%D1%82%D0%B0) системе). Подпись связана как с автором, так и с самим документом с помощью криптографических методов, и не может быть подделана с помощью обычного копирования.

В ходе данной лабораторной работы необходимо изучить теоретическую часть о электронной подписи и алгоритме ГОСТ 3410, на основании которых создать приложение, использующие данные технологии.

**2. Краткие теоретические сведения**

ЭЦП — это реквизит [электронного документа](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82), полученный в результате криптографического преобразования [информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) с использованием [закрытого ключа](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87) подписи и позволяющий проверить отсутствие искажения информации в электронном документе с момента формирования подписи (целостность), принадлежность подписи владельцу [сертификата ключа подписи](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D1%8E%D1%87%D0%B0) (авторство), а в случае успешной проверки подтвердить факт подписания электронного документа (неотказуемость).

ГОСТ 34.10-2018 (полное название: «ГОСТ 34.10-2018. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи», англ. «Information technology. Cryptographic data security. Signature and verification processes of electronic digital signature») — действующий [межгосударственный](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B3%D0%BE%D1%81%D1%83%D0%B4%D0%B0%D1%80%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82) [криптографический](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) [стандарт](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82), описывающий алгоритмы формирования и проверки [электронной цифровой подписи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C) реализуемой с использованием операций в группе точек эллиптической кривой, определенной над конечным простым полем.

Все стандарты семейства 34.10 основаны на [эллиптических кривых](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%BB%D0%B8%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%B2%D0%B0%D1%8F). Стойкость этих алгоритмов основывается на сложности вычисления дискретного логарифма в группе точек эллиптической кривой, а также на стойкости соотвествующей хеш-функции.

Цифровая подпись позволяет:

1. [Аутентифицировать](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) лицо, подписавшее сообщение;
2. Контролировать целостность сообщения;
3. Защищать сообщение от подделок.

Стандарты используют одинаковую схему формирования электронной цифровой подписи. Новые стандарты с 2012 года отличается наличием дополнительного варианта параметров схем соответствующего длине секретного ключа порядка 512 бит.

После подписывания сообщения М к нему дописывается цифровая подпись размером 512 или 1024 [бит](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) и текстовое поле. В текстовом поле могут содержаться, например, дата и время отправки или различные данные об отправителе:

Данный алгоритм не описывает механизм генерации параметров, необходимых для формирования подписи, а только определяет, каким образом на основании таких параметров получить цифровую подпись. Механизм генерации параметров определяется на месте в зависимости от разрабатываемой системы.

**3. Схема алгоритма**

# 

# 4. Результаты выполнения

# 

Рисунок 4.1. Исходные данные

# 

Рисунок 4.2. Результат работы программы

**5. Вывод**

В рамках данной лабораторной работы были рассмотрены цифровая подпись и ГОСТ 3410.

Для демонстрации работы описанных выше алгоритмов было разработано и реализовано приложение с использованием указанного выше алгоритма.

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

from codecs import getdecoder

from codecs import getencoder

from hashlib import md5

from os import urandom

class GostCurve(object):

def \_\_init\_\_(self, p, q, a, b, x, y, e=None, d=None):

self.p = p

self.q = q

self.a = a

self.b = b

self.x = x

self.y = y

self.e = e

self.d = d

r1 = self.y \* self.y % self.p

r2 = ((self.x \* self.x + self.a) \* self.x + self.b) % self.p

if r1 != self.pos(r2):

raise ValueError("Invalid parameters")

self.\_st = None

def pos(self, v):

if v < 0:

return v + self.p

return v

def \_add(self, p1x, p1y, p2x, p2y):

if p1x == p2x and p1y == p2y:

t = ((3 \* p1x \* p1x + self.a) \* \_modinvert(2 \* p1y, self.p)) % self.p

else:

tx = self.pos(p2x - p1x) % self.p

ty = self.pos(p2y - p1y) % self.p

t = (ty \* \_modinvert(tx, self.p)) % self.p

tx = self.pos(t \* t - p1x - p2x) % self.p

ty = self.pos(t \* (p1x - tx) - p1y) % self.p

return tx, ty

def exp(self, degree, x=None, y=None):

x = x or self.x

y = y or self.y

tx = x

ty = y

if degree == 0:

raise ValueError("Bad degree value")

degree -= 1

while degree != 0:

if degree & 1 == 1:

tx, ty = self.\_add(tx, ty, x, y)

degree = degree >> 1

x, y = self.\_add(x, y, x, y)

return tx, ty

def st(self):

if self.e is None or self.d is None:

raise ValueError("non twisted Edwards curve")

if self.\_st is not None:

return self.\_st

self.\_st = (

self.pos(self.e - self.d) \* \_modinvert(4, self.p) % self.p,

(self.e + self.d) \* \_modinvert(6, self.p) % self.p,

)

return self.\_st

def get\_curve():

return GostCurve(

p=\_bytes2long(\_hexdec("C0000000000000000000000000000000000000000000000000000000000003C7")),

q=\_bytes2long(\_hexdec("5fffffffffffffffffffffffffffffff606117a2f4bde428b7458a54b6e87b85")),

a=\_bytes2long(\_hexdec("C0000000000000000000000000000000000000000000000000000000000003c4")),

b=\_bytes2long(\_hexdec("2d06B4265ebc749ff7d0f1f1f88232e81632e9088fd44b7787d5e407e955080c")),

x=\_bytes2long(\_hexdec("0000000000000000000000000000000000000000000000000000000000000002")),

y=\_bytes2long(\_hexdec("a20e034bf8813ef5c18d01105e726a17eb248b264ae9706f440bedc8ccb6b22c")),

)

def get\_private\_key():

return \_bytes2long(urandom(32)[::-1])

def sign(curve, private\_key, text, block\_size=64):

h = md5(text.encode()).digest()

q = curve.q

e = \_bytes2long(h) % q

if e == 0:

e = 1

while True:

k = \_bytes2long(urandom(block\_size)) % q

if k == 0:

continue

c\_x, \_ = curve.exp(k)

r = c\_x % q

if r == 0:

continue

d = private\_key \* r

k \*= e

s = (d + k) % q

if s == 0:

continue

break

return \_long2bytes(s, block\_size) + \_long2bytes(r, block\_size)

def verify(curve, public\_key\_tuple, digest, signature, block\_size=64):

if len(signature) != block\_size \* 2:

raise ValueError("Invalid signature length")

q = curve.q

p = curve.p

s = \_bytes2long(signature[:block\_size])

r = \_bytes2long(signature[block\_size:])

if r <= 0 or r >= q or s <= 0 or s >= q:

return False

e = \_bytes2long(digest) % curve.q

if e == 0:

e = 1

v = \_modinvert(e, q)

z1 = s \* v % q

z2 = q - r \* v % q

p1x, p1y = curve.exp(z1)

q1x, q1y = curve.exp(z2, public\_key\_tuple[0], public\_key\_tuple[1])

c\_x = q1x - p1x

if c\_x < 0:

c\_x += p

c\_x = \_modinvert(c\_x, p)

z1 = q1y - p1y

c\_x = c\_x \* z1 % p

c\_x = c\_x \* c\_x % p

c\_x = c\_x - p1x - q1x

c\_x = c\_x % p

if c\_x < 0:

c\_x += p

c\_x %= q

return c\_x == r

def \_bytes2long(raw):

return int(\_hexenc(raw), 16)

def \_long2bytes(n, size=32):

res = hex(int(n))[2:].rstrip("L")

if len(res) % 2 != 0:

res = "0" + res

s = \_hexdec(res)

if len(s) != size:

s = (size - len(s)) \* b"\x00" + s

return s

def \_hexdec(data):

\_hexdecoder = getdecoder("hex")

return \_hexdecoder(data)[0]

def \_hexenc(data):

\_hexencoder = getencoder("hex")

return \_hexencoder(data)[0].decode("ascii")

def \_modinvert(a, n):

if a < 0:

return n - \_modinvert(-a, n)

t, newt = 0, 1

r, newr = n, a

while newr != 0:

quotinent = r // newr

t, newt = newt, t - quotinent \* newt

r, newr = newr, r - quotinent \* newr

if r > 1:

return -1

if t < 0:

t = t + n

return t