Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №6

**Цифровая подпись**

Выполнил:

Зеневич А. В.

Проверил:

Артемьев В. С.

Минск 2019

Содержание

[1. Постановка задачиError: Reference source not found](#_Toc506485975)2

2. Краткие теоретические сведения

3[. Результаты выполненияPAGEREF \_Toc506485977 \hError: Reference source not found](#_Toc506485977)

4[. ВыводыPAGEREF \_Toc506485978 \hError: Reference source not found](#_Toc506485978)

[ПРИЛОЖЕНИЕ 1. Исходный код программы.Error: Reference source not foundError: Reference source not foundPAGEREF \_Toc506485979 \hError: Reference source not found](#_Toc506485979)

# 1. Постановка задачи

1. Изучить теоретические сведения.

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

**2. Краткие теоретические сведения**

**Цифровая подпись** (ЦП) позволяет подтвердить авторство электронного документа (будь то реальное лицо или, например, аккаунт в [криптовалютной](https://ru.wikipedia.org/wiki/Криптовалюта) системе). Подпись связана как с автором, так и с самим документом с помощью криптографических методов, и не может быть подделана с помощью обычного копирования.

ЭЦП — это реквизит [электронного документа](https://ru.wikipedia.org/wiki/Электронный_документ), полученный в результате криптографического преобразования [информации](https://ru.wikipedia.org/wiki/Информация) с использованием [закрытого ключа](https://ru.wikipedia.org/wiki/Закрытый_ключ) 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/Сертификат_открытого_ключа) (авторство), а в случае успешной проверки подтвердить факт подписания электронного документа (неотказуемость).

## Основные принципы

Широко применяемая в настоящее время технология электронной подписи основана на [асимметричном шифровании с открытым ключом](https://ru.wikipedia.org/wiki/Криптосистема_с_открытым_ключом) и опирается на следующие принципы:

* Можно сгенерировать пару очень больших чисел (открытый ключ и закрытый ключ) так, чтобы, зная открытый ключ, нельзя было вычислить закрытый ключ за разумный срок. Механизм генерации ключей строго определён и является общеизвестным. При этом каждому открытому ключу соответствует определённый закрытый ключ. Если, например, Иван Иванов публикует свой открытый ключ, то можно быть уверенным, что соответствующий закрытый ключ есть только у него.
* Имеются надёжные методы шифрования, позволяющие зашифровать сообщение закрытым ключом так, чтобы расшифровать его можно было только открытым ключом. Механизм шифрования является общеизвестным.
* Если электронный документ поддается расшифровке с помощью открытого ключа, то можно быть уверенным, что он был зашифрован с помощью уникального закрытого ключа. Если документ расшифрован с помощью открытого ключа Ивана Иванова, то это подтверждает его авторство: зашифровать данный документ мог только Иванов, т.к. он является единственным обладателем закрытого ключа.

Однако шифровать весь документ было бы неудобно, поэтому шифруется только его [хеш](https://ru.wikipedia.org/wiki/Хеширование) - небольшой объём данных, жёстко привязанный к документу с помощью математических преобразований и идентифицирующий его. Шифрованный хеш и является электронной подписью.

**ГОСТ 3410** (полное название: «*ГОСТ 3410. 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/Межгосударственный_стандарт) [криптографический](https://ru.wikipedia.org/wiki/Криптография) [стандарт](https://ru.wikipedia.org/wiki/Стандарт), описывающий алгоритмы формирования и проверки [электронной цифровой подписи](https://ru.wikipedia.org/wiki/Цифровая_подпись) реализуемой с использованием операций в группе точек эллиптической кривой, определенной над конечным простым полем.

Цифровая подпись позволяет:

1. [Аутентифицировать](https://ru.wikipedia.org/wiki/Аутентификация) лицо, подписавшее сообщение;
2. Контролировать целостность сообщения;
3. Защищать сообщение от подделок;

## Алгоритм:

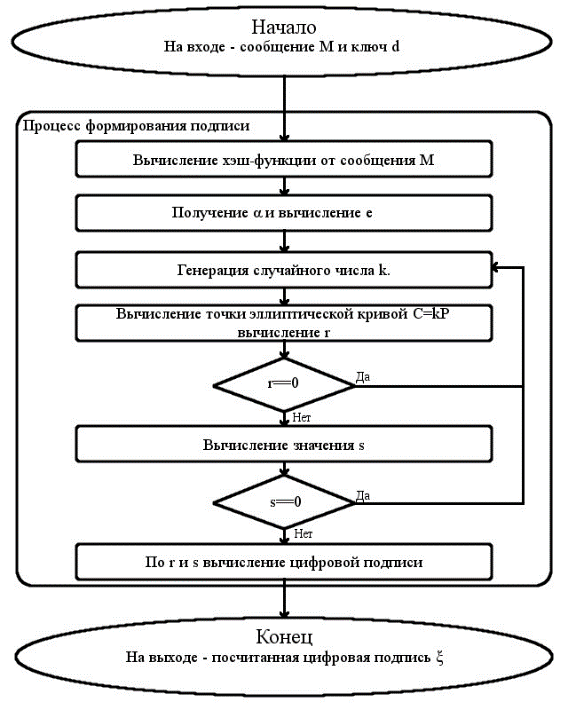
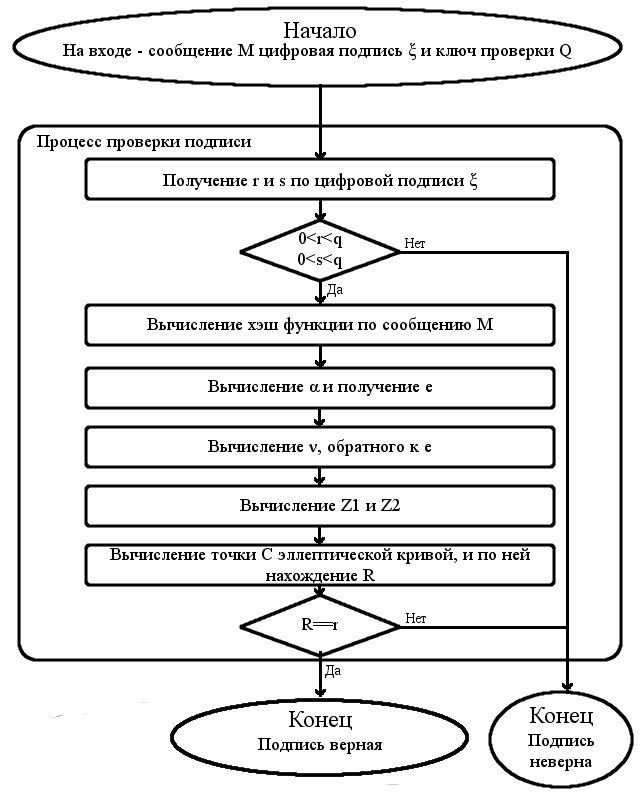
Приводится описание варианта схемы ЭЦП с длиной секретного ключа 256 бит. Для секретных ключей длиной 512 бит (второй вариант формирования ЭЦП, описанный в стандарте) все преобразования аналогичны.

### Двоичные векторы:

Между двоичными [векторами](https://ru.wikipedia.org/wiki/Вектор) длины 256 {\displaystyle {\bar {h}}=(\alpha \_{255},...,\alpha \_{0})} и целыми числами {\displaystyle z\leq 2^{256}} ставится [взаимно-однозначное соответствие](https://ru.wikipedia.org/wiki/Биекция) по следующему правилу {\displaystyle z=\sum \_{i=0}^{255}\alpha \_{i}2^{i}}. Здесь {\displaystyle \alpha \_{i}}либо равно 0, либо равно 1. Другими словами {\displaystyle {\bar {h}}} — это представление числа z в двоичной системе счисления.

Результатом операции конкатенации двух векторов {\displaystyle {\bar {h\_{1}}}=(\alpha \_{255},...,\alpha \_{0})} и {\displaystyle {\bar {h\_{2}}}=(\beta \_{255},...,\beta \_{0})} называется вектор длины 512 {\displaystyle ({\bar {h\_{1}}}|{\bar {h\_{2}}})=(\alpha \_{255},...,\alpha \_{0},\beta \_{255},...,\beta \_{0})}. Обратная операция — операция разбиения одного вектора длины 512 на два вектора длины 256.

Алгоритм:



# Выводы

# В результате выполнения лабораторной работы была получена реализация алгоритма ГОСТ3410, а также алгоритм генерации ключей для данного алгоритма. Данный алгоритм относится к семейству асимметричных алгоритмов.

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

from os import urandom

from codecs import getdecoder

from codecs import getencoder

\_hexdecoder = getdecoder("hex")

\_hexencoder = getencoder("hex")

def hexdec(data):

return \_hexdecoder(data)[0]

def hexenc(data):

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

def modinvert(a, n):

if a < 0:

# k^-1 = p - (-k)^-1 mod p

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

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

class GOST3410Curve(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:

# double

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):

"""Compute s/t parameters for twisted Edwards curve points conversion

"""

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 public\_key(curve, prv):

""" Generate public key from the private one"""

return curve.exp(prv)

def sign(curve, prv, digest):

""" Calculate signature for provided digest"""

size = 64

q = curve.q

e = bytes2long(digest) % q

if e == 0:

e = 1

while True:

k = bytes2long(urandom(size)) % q

if k == 0:

continue

r, \_ = curve.exp(k)

r %= q

if r == 0:

continue

d = prv \* r

k \*= e

s = (d + k) % q

if s == 0:

continue

break

return long2bytes(s, size) + long2bytes(r, size)

def verify(curve, pub, digest, signature):

""" Verify provided digest with the signature"""

size = 64

if len(signature) != size \* 2:

raise ValueError("Invalid signature length")

q = curve.q

p = curve.p

s = bytes2long(signature[:size])

r = bytes2long(signature[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, pub[0], pub[1])

lm = q1x - p1x

if lm < 0:

lm += p

lm = modinvert(lm, p)

z1 = q1y - p1y

lm = lm \* z1 % p

lm = lm \* lm % p

lm = lm - p1x - q1x

lm = lm % p

if lm < 0:

lm += p

lm %= q

# This is not constant time comparison!

return lm == r

def prv\_unmarshal(prv):

"""Unmarshal private key"""

return bytes2long(prv[::-1])

def pub\_marshal(pub):

"""Marshal public key"""

size = 32

return (long2bytes(pub[1], size) + long2bytes(pub[0], size))[::-1]

def pub\_unmarshal(pub):

"""Unmarshal public key"""

size = 32

pub = pub[::-1]

return (bytes2long(pub[size:]), bytes2long(pub[:size]))

def uv2xy(curve, u, v):

"""Convert twisted Edwards curve U,V coordinates to Weierstrass X,Y"""

s, t = curve.st()

k1 = (s \* (1 + v)) % curve.p

k2 = curve.pos(1 - v)

x = t + k1 \* modinvert(k2, curve.p)

y = k1 \* modinvert(u \* k2, curve.p)

return x % curve.p, y % curve.p

def xy2uv(curve, x, y):

"""Convert Weierstrass X,Y coordinates to twisted Edwards curve U,V"""

s, t = curve.st()

xmt = curve.pos(x - t)

u = xmt \* modinvert(y, curve.p)

v = curve.pos(xmt - s) \* modinvert(xmt + s, curve.p)

return u % curve.p, v % curve.p