Учреждение образования

Белорусский государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по лабораторной работе №6

**Цифровая подпись**

|  |  |
| --- | --- |
| Выполнил:  студент группы 653501  Ковалевская Алина | Проверил:  Артемьев В.С. |

Минск, 2019

**Задача**

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

**Введение**

ГОСТ Р 34.10-2012 и ГОСТ Р 34.10-2001 основаны на [эллиптических кривых](https://ru.wikipedia.org/wiki/Эллиптическая_кривая). Стойкость этих алгоритмов основывается на сложности вычисления дискретного логарифма в группе точек эллиптической кривой, а также на стойкости хэш-функции. Для ГОСТ Р 34.10-2012 используется хэш-функция по [ГОСТ Р 34.11-2012](https://ru.wikipedia.org/wiki/ГОСТ_Р_34.11-2012). Для ГОСТ Р 34.10-2001 — [ГОСТ Р 34.11-94](https://ru.wikipedia.org/wiki/ГОСТ_Р_34.11-94).

Стандарт ГОСТ Р 34.10-2012 использует ту же схему формирования электронной цифровой подписи, что и ГОСТ Р 34.10-2001. Новый стандарт отличается наличием дополнительного варианта параметров схем (соответствующего длине секретного ключа порядка 512 бит) и требованием использования функций хэширования ГОСТ Р 34.11-2012: первый вариант требований к параметрам (такой же, как в ГОСТ Р 34.10-2001, соответствующий длине секретного ключа порядка 256 бит) предусматривает использование хэш-функции с длиной хэш-кода 256 бит, дополнительный вариант требований к параметрам предусматривает использование хэш-функции с длиной хэш-кода 512 бит.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | Сообщение М | | +   |  |  | | --- | --- | | Цифровая подпись | Текст | |  |
|  |  | Дополнение |

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.

**Алгоритм**

**Алгоритм выработки подписи**

Процедура подписи сообщения включает в себя следующие этапы:

1.Вычислить h(M){\displaystyle h(M)} -значение хеш-функции h{\displaystyle h} от сообщения M{\displaystyle M}. Если h(M)(mod q)=0{\displaystyle h(M)(mod ~q)=0}, присвоить h(M){\displaystyle h(M)} значение 0255||1{\displaystyle 0^{255} ||1}.

2. Выработать целое число k{\displaystyle k}, такое, что 0<k<q{\displaystyle 0<k < q}.

3. Вычислить два значения:r=ak(mod p){\displaystyle r=a^k(mod ~p)} и r′=r(mod q){\displaystyle r' = r (mod ~q)}. Если r′=0{\displaystyle r' =0}, перейти к этапу 2 и выработать другое значение числа k{\displaystyle k}.

4. С использованием секретного ключа x{\displaystyle x} пользователя (отправителя сообщения) вычислить значение s=(xr′+kh(M))(mod q){\displaystyle s= (xr' + kh(M))(mod ~q)}. Если s=0{\displaystyle s=0}, перейти к этапу 2, в противном случае закончить работу алгоритма.

Подписью сообщения М является вектор <r′>256||<s>256{\displaystyle <r'>\_{256} || <s>\_{256} }.

**Процедура проверки подписи**

Получатель должен проверить подлинность сообщения и подлинность ЭЦП, осуществляя ряд операций (вычислений). Это возможно при наличии у получателя открытого ключа отправителя, пославшего сообщение, Процедура проверки включает в себя следующие этапы:

1. Проверить условие: 0<s<q{\displaystyle 0< s < q} и 0<r′<q{\displaystyle 0 < r'< q}. Если хотя бы одно из этих условий не выполнено, то подпись считается недействительной.

2. Вычислять h(M1){\displaystyle h(M\_1 )} -значение хеш-функции h{\displaystyle h} от полученного сообщения M1{\displaystyle M\_{1}} . Если H(M1)(mod q)=0{\displaystyle H(M\_1 )(mod ~q)=0}, присвоить h(M1){\displaystyle h(M1 )} значение 0255||1{\displaystyle 0^{255} || 1}.

3. Вычислить значение v=(h(M1))q−2(mod q){\displaystyle v= (h(M\_1 ))^{q-2} (mod ~q)}

4. Вычислить значения: z1=sv(mod q){\displaystyle z\_1 = sv (mod ~q)} и z2=(q−r′)v(mod q){\displaystyle z\_2 = (q-r' ) v (mod ~q)}

5. Вычислить значение u=(as1ys2(modp))(modq){\displaystyle u = (a^{s1} y^{s2} (mod p)) (mod q)}

6. Проверить условие: r′=u{\displaystyle r' = u}.

При совпадении значений r{\displaystyle r} и u{\displaystyle u} получатель принимает решение о том, что полученное сообщение подписано данным отправителем и в процессе передачи не нарушена целостность сообщения, т.е. M=M{\displaystyle M=M}. В противном случае подпись считается недействительной.

**Алгоритм**

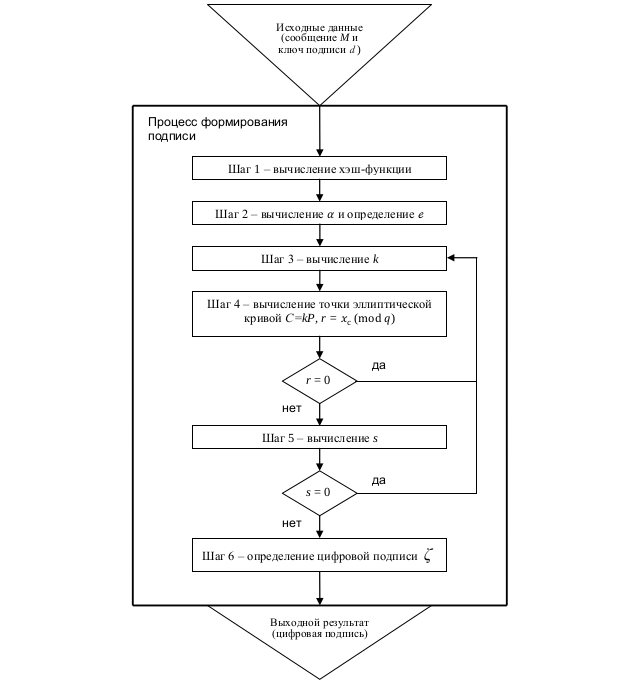


Рисунок 1 Блок-схема алгоритма подписи

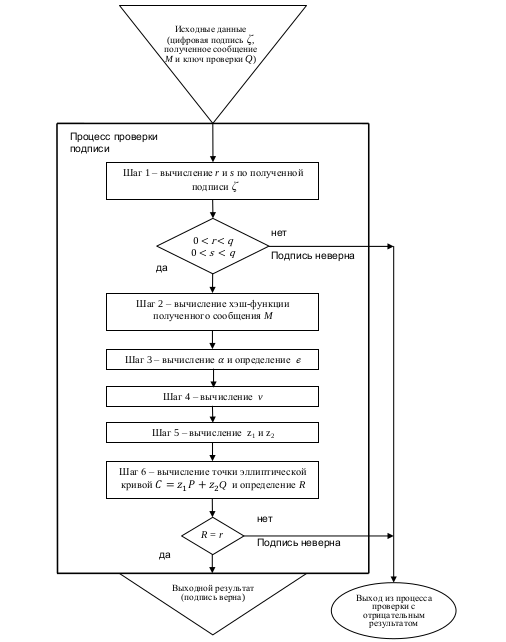
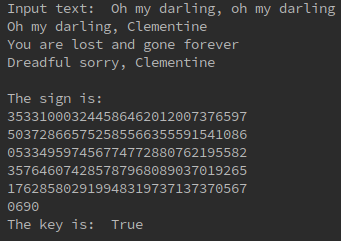


Рисунок 2 Блок-схема алгоритма проверки подписи

**Демонстрация работы программы**



**Вывод**

В ходе написания лабораторной работы были изучены алгоритмы создания и проверки электронной цифровой подписи ГОСТ 3410, а также написаны их программные реализации. Были получены навыки усложнения и увеличения криптостойкости алгоритма создания электронной цифровой подписи, а также изучены модификации работы алгоритма ГОСТ 3410.

**Программный код**

# from pygost import gost3411 # use this realization, if smth fails

import random

import sys

from elliptic\_curve import EllipticCurvePoint

def initial\_data():

p = 0x8000000000000000000000000000000000000000000000000000000000000431

a = 0x7

b = 0x5FBFF498AA938CE739B8E022FBAFEF40563F6E6A3472FC2A514C0CE9DAE23B7E

m = 0x8000000000000000000000000000000150FE8A1892976154C59CFC193ACCF5B3

q = 0x8000000000000000000000000000000150FE8A1892976154C59CFC193ACCF5B3

x\_p = 0x2

y\_p = 0x8E2A8A0E65147D4BD6316030E16D19C85C97F0A9CA267122B96ABBCEA7E8FC8

x\_q = 0x7F2B49E270DB6D90D8595BEC458B50C58585BA1D4E9B788F6689DBD8E56FD80B

y\_q = 0x26F1B489D6701DD185C8413A977B3CBBAF64D1C593D26627DFFB101A87FF77DA

if not is\_valid\_q(q, m):

raise ValueError('q is not valid')

return p, a, b, m, q, x\_p, y\_p, x\_q, y\_q

def nsplit(s, n): # Split a list into sublists of size "n"

return [s[k:k + n] for k in range(0, len(s), n)]

def string\_to\_bit\_array(raw\_text): # Convert a string into a list of bits

array = list()

for char in raw\_text:

binval = binvalue(char, 8) # Get the char value on one byte

array.extend([int(x) for x in list(binval)]) # Add the bits to the final list

return array

def bit\_array\_to\_string(array): # Recreate the string from the bit array

res = ''.join([chr(int(y, 2)) for y in [''.join([str(x) for x in \_bytes]) for \_bytes in nsplit(array, 8)]])

return res

def binvalue(val, bitsize): # Return the binary value as a string of the given size

binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]

if len(binval) > bitsize:

raise Exception("binary value larger than the expected size")

while len(binval) < bitsize:

binval = "0" + binval # Add as many 0 as needed to get the wanted size

return binval

def int\_to\_bit(n): # convert int to array of bits

return [int(digit) for digit in bin(n)[2:]]

def bit\_to\_int(value): # convert bit array to int

return int(''.join(map(str, value)), 2)

def inverse(num, modulo): # \*\* -1

def gcd\_extended(a, b):

if a == 0:

return b, 0, 1

divider, x, y = gcd\_extended(b % a, a)

return divider, y - (b // a) \* x, x

gcd, x, \_ = gcd\_extended(num, modulo)

if gcd == 1:

return x % modulo

raise ValueError("invalid arguments")

def generate\_gost\_digital\_signature(d, M):

p, a, b, m, q, x\_p, y\_p, x\_q, y\_q = initial\_data()

if not is\_valid\_d(d, q):

raise ValueError('d is not correct')

# step 1

l = get\_l(q)

h\_ = get\_stribog\_hash(M, l)

# step 2

alpha = int(h\_, 16)

e = alpha % q

e = 1 if e == 0 else e

# step 3

r = 0

s = 0

while True:

k = random.randint(0, q)

# k = 0x77105C9B20BCD3122823C8CF6FCC7B956DE33814E95B7FE64FED924594DCEAB3

# step 4

# C = k \* P

P = EllipticCurvePoint([x\_p, y\_p], [a, b, p])

C = k \* P

r = C.x % q

if r == 0:

continue

# step 5

s = (r \* d + k \* e) % q

if s == 0:

continue

else:

break

# step 6

# sign = str(r) + str(s)

sign = (r, s)

return sign

def check\_gost\_digital\_signature(sign, M, d):

p, a, b, m, q, x\_p, y\_p, x\_q, y\_q = initial\_data()

# step 1

# r, s = nsplit(sign, len(sign) // 2)

r, s = sign

if not (0 < r < q and 0 < s < q):

raise ValueError('Sign is not correct!')

# step 2

l = get\_l(q)

h\_ = get\_stribog\_hash(M, l)

# step 3

alpha = int(h\_, 16)

e = alpha % q

e = 1 if e == 0 else e

# step 4

# v = (e \*\* (-1)) % q

v = inverse(e, q)

# step 5

z1 = (s \* v) % q

z2 = (-r \* v) % q

# step 6

# C = z1 \* P + z2 \* Q

P = EllipticCurvePoint([x\_p, y\_p], [a, b, p])

Q = EllipticCurvePoint([x\_q, y\_q], [a, b, p])

# if not is\_valid\_Q(Q, P, d):

# raise ValueError('Q is not correct')

P = z1 \* P

Q = z2 \* Q

C = P + Q

R = C.x % q

# step 7

if r == R:

return True

else:

print('Sign is not correct, R != r')

return False

# raise ValueError('Sign is not correct, R != r')

def is\_valid\_q(q, m):

n = m / q

if not (m == n \* q and n >= 1 and n % 1 == 0):

return False

if (2 \*\* 254 < q < 2 \*\* 256) or (2 \*\* 508 < q < 2 \*\* 512):

return True

return False

def is\_valid\_d(d, q):

return 0 < d < q

def is\_valid\_Q(Q, P, d):

return Q == d \* P

def is\_valid\_P(P, q, O):

return P != O and q \* P == O

def validate\_params(p, q, B, m, E):

if m == p:

return False

if J(E) == 0 or J(E) == 1728:

return False

if B >= 31:

for t in range(B):

if p \*\* t == 1 % q:

return False

return True

def J(E):

# curve invariant

pass

def get\_l(q):

if 2 \*\* 254 < q < 2 \*\* 256:

return 256

elif 2 \*\* 508 < q < 2 \*\* 512:

return 512

else:

return None

def get\_stribog\_hash(message, length):

import stribog # gost 3411

return stribog.entry().FromString(message, length)

def main():

# M = 'Oh my darling, oh my darling' \

# 'Oh my darling, Clementine' \

# 'You are lost and gone forever' \

# 'Dreadful sorry, Clementine'

if len(sys.argv) < 2:

exit(1)

file\_in = sys.argv[1]

file\_out = sys.argv[2]

with open(file\_in, 'r+') as f:

M = f.read()

print 'Input text: ', M

d = 0x7A929ADE789BB9BE10ED359DD39A72C11B60961F49397EEE1D19CE9891EC3B28

# use for this lab only python 2.7 (stribog works only in pythno 2.7)

try:

sign = generate\_gost\_digital\_signature(d, M)

except Exception as ex:

print ex

sys.exit(-1)

print 'The sign is: '

splitted\_sign = ''.join([str(i) for i in sign])

n = 30

splitted\_sign = [splitted\_sign[i:i+n] for i in range(0, len(splitted\_sign), n)]

for i in splitted\_sign:

print i

with open(file\_out, 'w+') as f:

f.write(''.join([str(i) for i in sign]))

# Q = key of check

try:

validation = check\_gost\_digital\_signature(sign, M, d)

except Exception as ex:

print ex

sys.exit(-1)

print 'The key is: ', validation

if \_\_name\_\_ == '\_\_main\_\_':

main()