Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

Институт компьютерных наук и кибербезопасности

Высшая школа кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**«Протокол электронной цифровой подписи ГОСТ Р 34.10-2018»**

по дисциплине «Криптографические методы защиты информации»

Выполнил

студент гр. 5151004/90101 Кондачков Е.Д.

Преподаватель

Ассистент Ярмак А.В.

Санкт-Петербург

2023

Содержание

**Элементы оглавления не найдены.**

# Цель работы

Изучение протокола электронной цифровой подписи   
ГОСТ Р 34.10-2018, безопасность которого основана на задаче дискретного логарифмирования в группе точек эллиптической кривой.

# Задачи работы

Согласно варианту разработать программу, реализующую криптоситему, поддерживающую формирование и проверку подписи по алгоритму ГОСТ Р 34.10-2018, а также допускать возможность использования различных ключей.

# Ход работы

В ходе данной работы был реализована программа, позволяющая подписывать и проверять электронную цифровую подпись по алгоритму ГОСТ Р 34.10-2018. Для выполнения работы был получен вариант задания от преподавателя.

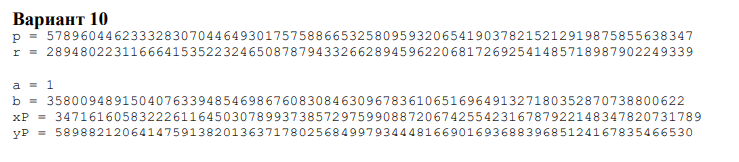


Рисунок 1 – Вариант задания

Для проверки программы использовался файл с простейшим текстом

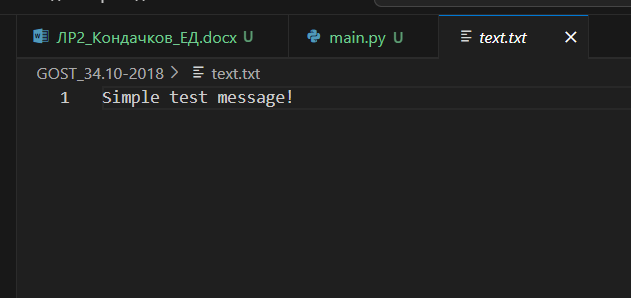


Рисунок 2 – Тестовый файл

Для формирования подписи были сгенерированы дополнительные параметры: ключ шифрования d и точка Q – использующаяся в качестве ключа расшифрования (Q = d\*P).

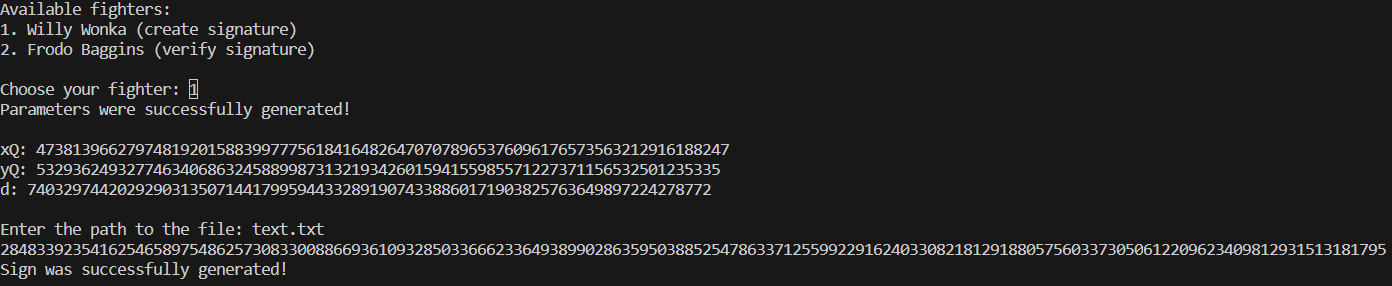


Рисунок 3 – Сгенерированные параметры и хеш сообщения.

По всем указанным данным была получена ЭПЦ.

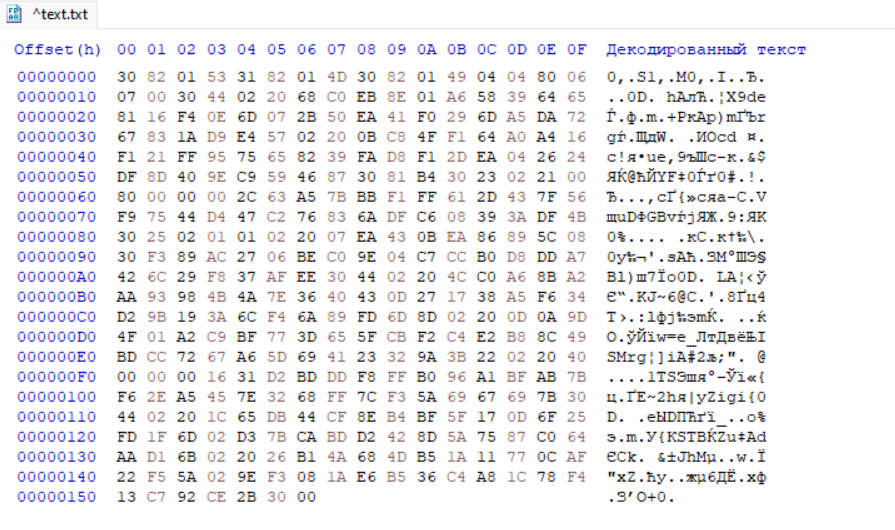


Рисунок 4 – Сформированная подпись в шестнадцатиричном виде

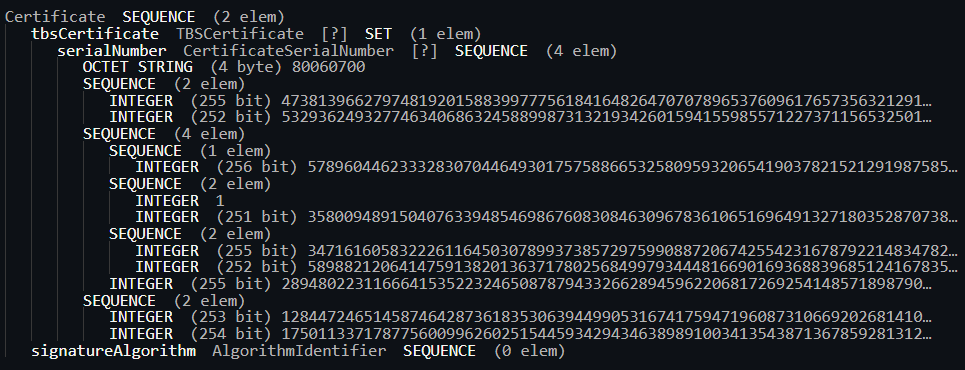


Рисунок 5 – Расшифрованная подпись

# Ответы на контрольные вопросы

1. Перечислите преимущества криптосистем на эллиптических кривых по сравнению с другими криптосистемами.

* На эллиптических кривых сложность алгоритма дискретного логарифмирования намного выше;
* Криптосистемы на эллиптических кривых для представления ключа требуют меньше бит, что приводит к упрощению многих операций, а также снижению требований к системе.

1. Почему в стандарте ГОСТ Р 34.10-2018 введено требование ?

Если 𝐸(𝔽𝑝) = 𝑝, где 𝐸(𝔽𝑝) – множество точек. Если p простое, то это означает, что порядок эллиптической кривой является простым числом и может быть уязвимой для атаки по сведению задачи дискретного логарифмирования к вычислению квадратного корня.

1. Если нарушитель имеет возможность обращать хэш-функцию, как он может подделать сообщение и подпись?

Может изменить исходное сообщение, вычислить новый хэш и подписать своим приватным ключом. В таком случае, при проверке подписи получатель будет считать подпись действительной, так как хэш соответствует исходному сообщению.

1. Почему случайный показатель k не должен повторяться в течение времени жизни ключа?

Случайность показателя k является гарантом уникальности значений подписи и её надежности.

# Выводы по работе

Была разработана программа реализующая алгоритм подписи   
ГОСТ Р 34.10 – 2018, получены знания об устройстве механизма подписи на основе дискретного логарифмирования на эллиптических кривых и изучены преимущества данной схемы.

# Приложение А

Листинг разработанной программы на языке Python3:

from asn import ASN

from Crypto.Util.number import inverse

from random import randint

import numpy as np

import gostcrypto

def get\_points\_sum(x1, y1, x2, y2, A = 1,

p = 57896044623332830704464930175758866532580959320654190378215212919875855638347):

lamda = 0

# Если складываем с бесконечно удаленной точкой

if x1 == 0 and y1 == 0:

return x2, y2

elif x2 == 0 and y2 == 0:

return x1, y1

# Если складываем P = (a , b) и P = (a, -b)

elif x1 == x2 and y1 == -y2:

return 0, 0 # бесконечно удаленная точка

# Если P = Q

elif x1 == x2 and y1 == y2:

lamda = ((3 \* (x1\*\*2) + A) \* inverse(2 \* y1, p)) % p

# Если P != Q

else:

lamda = ((y2 - y1) \* inverse((x2 - x1), p)) % p

x3 = (lamda\*\*2 - x1 - x2) % p

y3 = (lamda \* (x1 - x3) - y1) % p

return x3, y3

def get\_binary\_vector(x):

lst = []

while(x != 0):

lst.append(x % 2)

x //= 2

vctr = np.array(lst[::-1])

return vctr

def multiply\_binary(P\_x, P\_y, k):

if(k == 0):

return 0, 0 # бесконечно удаленная точка

elif(k == 1):

return P\_x, P\_y

a = get\_binary\_vector(k)

Q\_x, Q\_y = 0, 0

for i in a:

Q\_x, Q\_y = get\_points\_sum(Q\_x, Q\_y, Q\_x, Q\_y)

if(i == 1):

Q\_x, Q\_y = get\_points\_sum(Q\_x, Q\_y, P\_x, P\_y)

return Q\_x, Q\_y

def generate\_parameters(q, P\_x, P\_y):

d = randint(1, q - 1)

Q\_x, Q\_y = multiply\_binary(P\_x, P\_y, d)

print("Parameters were successfully generated!\n")

print("xQ:", Q\_x, "\nyQ:", Q\_y, "\nd:", d, "\n")

return Q\_x, Q\_y, d

def get\_text(filename = None):

if filename == None:

filename = input("Enter the path to the file: ")

text = b''

with open(filename, "rb") as file:

for line in file:

text += line

return text, filename

#------------------CREATING BLOCK------------------#

def form\_sign(Q\_x, Q\_y, A, B, p, q, P\_x, P\_y, d, text):

hash = gostcrypto.gosthash.new('streebog256', data=text)

hash = hash.hexdigest()

hash = hash.encode('utf-8')

hash = int.from\_bytes(hash, "big")

e = hash % q

if(e == 0):

e = 1

while True:

k = randint(1, q - 1)

C\_x, C\_y = multiply\_binary(P\_x, P\_y, k)

r = C\_x % q

if(r == 0):

continue

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

if(s != 0):

break

sign = []

sign.append(Q\_x)

sign.append(Q\_y)

sign.append(p)

sign.append(A)

sign.append(B)

sign.append(P\_x)

sign.append(P\_y)

sign.append(q)

sign.append(r)

sign.append(s)

print("Sign was successfully generated!\n")

return sign

def generate\_signature(Q\_x, Q\_y, A, B, p, q, P\_x, P\_y, d):

text, filename = get\_text()

sign = form\_sign(Q\_x, Q\_y, A, B, p, q, P\_x, P\_y, d, text)

asn\_text = ASN.encrypt\_gost\_eds(sign)

with open("^" + filename, "wb") as enc:

enc.write(asn\_text)

#------------------VERIFYING BLOCK------------------#

def check\_sign(q, Q\_x\_dec, Q\_y\_dec, P\_x\_dec, P\_y\_dec, q\_dec, r\_dec, s\_dec, text):

hash = gostcrypto.gosthash.new('streebog256', data=text)

hash = hash.hexdigest()

hash = hash.encode('utf-8')

a = int.from\_bytes(hash, "big")

e = a % q

if(e == 0):

e = 1

v = inverse(e, q)

z\_1 = (s\_dec \* v) % q

z\_2 = ((-1) \* r\_dec \* v) % q

P\_x\_dec, P\_y\_dec = multiply\_binary(P\_x\_dec, P\_y\_dec, z\_1)

Q\_x\_dec, Q\_y\_dec = multiply\_binary(Q\_x\_dec, Q\_y\_dec, z\_2)

C\_x, C\_y = get\_points\_sum(P\_x\_dec, P\_y\_dec, Q\_x\_dec, Q\_y\_dec)

R = C\_x % q

print("R:", R, "\nr:", r\_dec, "\n")

if(R == r\_dec):

print("The signature is valid")

else:

print("The signature is invalid")

def verify\_signature(q):

text, filename = get\_text()

sign\_asn, \_ = get\_text('^' + filename)

Q\_x\_dec, Q\_y\_dec, P\_x\_dec, P\_y\_dec, A\_dec, B\_dec, p\_dec, q\_dec, r\_dec, s\_dec = ASN.decrypt\_gost\_eds(sign\_asn)

check\_sign(q, Q\_x\_dec, Q\_y\_dec, P\_x\_dec, P\_y\_dec, q\_dec, r\_dec, s\_dec, text)

def main():

p = 57896044623332830704464930175758866532580959320654190378215212919875855638347

q = 28948022311666415352232465087879433266289459622068172692541485718987902249339

A = 1

B = 3580094891504076339485469867608308463096783610651696491327180352870738800622

xP = 34716160583222611645030789937385729759908872067425542316787922148347820731789

yP = 5898821206414759138201363717802568499793444816690169368839685124167835466530

while True:

print('''

Warning: You will not be able to decrypt the file or verify

the signature if the action was not performed at the current start of the program.

Available fighters:

1. Willy Wonka (create signature)

2. Frodo Baggins (verify signature)

''' )

command = input("Choose your fighter: ")

if command == '1' or command == 'Willy Wonka':

Q\_x, Q\_y, d = generate\_parameters(q, xP, yP)

generate\_signature(Q\_x, Q\_y, A, B, p, q, xP, yP, d)

elif command == '2' or command == 'Frodo Baggins':

verify\_signature(q)

else:

print("incorrect input, try again")

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

main()