Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЕТ

к лабораторной работе №6

на тему

**Цифровая подпись.**

Выполнил Д.С. Шевцова

Проверил Герчик А. В.

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc158758843)

[1 Краткие теоретические сведения 4](#_Toc158758844)

[2 Результаты выполнения лабораторной работы](#_Toc158758845) 6

[Заключение](#_Toc158758846) 7

[Приложение А](#_Toc158758848) [(обязательное)](#_Toc158758849) [Листинг кода](#_Toc158758850) 8

## ВВЕДЕНИЕ

В данной лабораторной работе нужно реализовать программное средство формирования и проверки ЭЦП на базе алгоритма ГОСТ 34.10.

## 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Гост 34.10-2012, известный как "Гост 34.10", представляет собой стандарт, который определяет алгоритм цифровой подписи, разработанный для обеспечения безопасности и целостности данных. Он является частью российской криптографической стандартизации и обеспечивает методы создания и проверки цифровых подписей.

Алгоритм Гост 34.10 использует асимметричную криптографию, что означает, что для создания и проверки подписи используются разные ключи: приватный и публичный. Приватный ключ используется для подписания сообщения, а публичный ключ — для его проверки. Алгоритм основывается на математических принципах, связанных с конечными полями и эллиптическими кривыми.

Процесс создания цифровой подписи включает несколько ключевых этапов. Сначала происходит генерация ключей, в ходе которой создается пара ключей — приватный и публичный. Приватный ключ должен храниться в секрете, в то время как публичный ключ доступен для всех желающих. Далее следует хеширование сообщения: перед подписью сообщение проходит через хеш-функцию, которая преобразует его в фиксированной длины хэш. Этот хэш является уникальным представлением данных сообщения. Затем осуществляется создание подписи, на этом этапе полученный хэш шифруется с использованием приватного ключа. Результат этого процесса — цифровая подпись, которая будет передана вместе с оригинальным сообщением. Наконец, отправитель отправляет как оригинальное сообщение, так и его цифровую подпись.

Процесс проверки подписи начинается с получения сообщения и подписи: получатель получает оригинальное сообщение и соответствующую ему подпись. Далее следует хеширование полученного сообщения: получатель повторно применяет ту же хеш-функцию к полученному сообщению для получения нового хэша. Затем происходит расшифровка подписи: получатель расшифровывает цифровую подпись, используя публичный ключ отправителя. Это позволяет получить оригинальный хэш, который был создан при подписании. Наконец, осуществляется сравнение хэшей: если новый хэш, полученный от сообщения, совпадает с хэшем, полученным из подписи, это означает, что сообщение не было изменено и было действительно подписано отправителем.

Гост 34.10 используется в различных сферах, включая электронные правительственные услуги, электронную коммерцию и другие системы, требующие надежной защиты данных и идентификации пользователей. Он стал важным инструментом в сфере информационной безопасности и помогает поддерживать доверие в электронных взаимодействиях. Также является ключевым стандартом для цифровых подписей в России, обеспечивая надежную защиту и аутентификацию данных. Его применение позволяет гарантировать целостность и подлинность сообщений в условиях современных угроз безопасности.

Ниже на рисунке 1.1 представлена блок схема алгоритма.

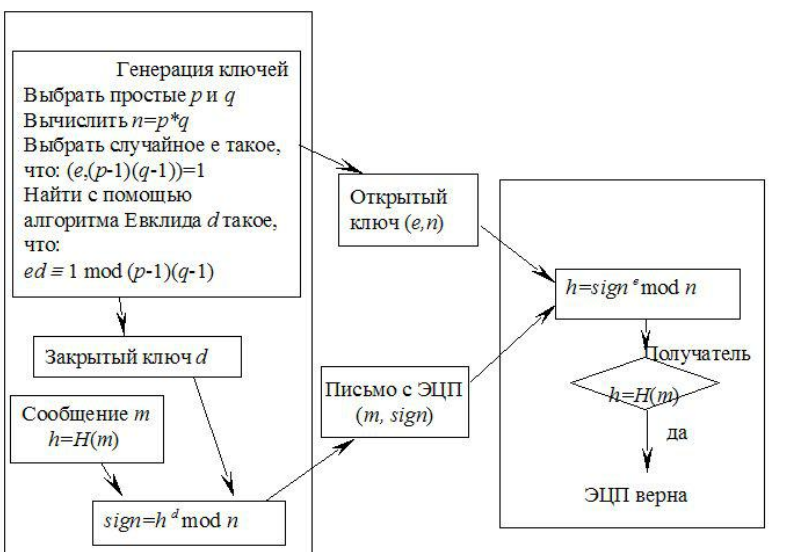


Рисунок 1.1 – Блок схема алгоритма

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Для реализации цифровой подписи используется алгоритм шифрования ГОСТ 34.11. Процесс начинается с генерации пары ключей — приватного и публичного. Приватный ключ хранится в секрете, а публичный доступен для всех. Затем сообщение, которое требуется подписать, проходит через хеш-функцию ГОСТ, преобразующую его в фиксированной длины хэш. Этот хэш шифруется с использованием приватного ключа, в результате чего создается цифровая подпись. Оригинальное сообщение и соответствующая ему цифровая подпись отправляются получателю.

После получения сообщения и подписи получатель выполняет проверку. Сначала он повторно применяет ту же хеш-функцию к полученному сообщению для получения нового хэша. Затем получатель расшифровывает цифровую подпись, используя публичный ключ отправителя, чтобы получить оригинальный хэш. В конце сравниваются оба хэша: если новый хэш совпадает с хэшем, полученным из подписи, это подтверждает, что сообщение не было изменено и действительно подписано отправителем.

На рисунке 2.1 показан результат выполнения программы по созданию цифровой подписи.

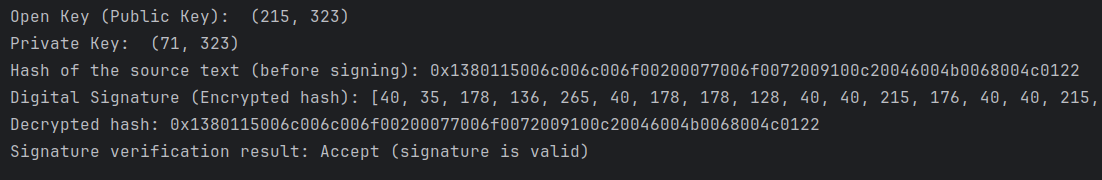


Рисунок 2.1 – Результат выполнения программы

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы была написана реализация программного средства по формированию и проверке ЭЦП на базе алгоритма ГОСТ 34.10.

# **ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

digital\_signature.py

import random

import gost

def co\_prime(a, b):

while b != 0:

a, b = b, a % b

return a

def extended\_gcd(aa, bb):

lastremainder, remainder = abs(aa), abs(bb)

x, lastx, y, lasty = 0, 1, 1, 0

while remainder:

lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)

x, lastx = lastx - quotient \* x, x

y, lasty = lasty - quotient \* y, y

return lastremainder, lastx \* (-1 if aa < 0 else 1), lasty \* (-1 if bb < 0 else 1)

def mod\_inv(a, m):

g, x, y = extended\_gcd(a, m)

if g != 1:

raise Exception('Modular inverse does not exist')

return x % m

def is\_prime(num):

if num == 2:

return True

if num < 2 or num % 2 == 0:

return False

for n in range(3, int(num \*\* 0.5) + 2, 2):

if num % n == 0:

return False

return True

def generate\_key\_pair(p, q):

if not (is\_prime(p) and is\_prime(q)):

raise ValueError('Both numbers must be prime.')

elif p == q:

raise ValueError('p and q cannot be equal')

n = p \* q

phi = (p - 1) \* (q - 1)

e = random.randrange(1, phi)

g = co\_prime(e, phi)

while g != 1:

e = random.randrange(1, phi)

g = co\_prime(e, phi)

d = mod\_inv(e, phi)

# Public key is (e, n) and private key is (d, n)

return (e, n), (d, n)

def encrypt(private\_key, plain\_text):

key, n = private\_key

cipher = [pow(ord(char), key, n) for char in plain\_text]

return cipher

def decrypt(public\_key, cipher\_text):

key, n = public\_key

plain = [chr(pow(char, key, n)) for char in cipher\_text]

return ''.join(plain)

def hash\_function(source\_text):

bin\_text = [hex(ord(elem)) for elem in source\_text]

blocks, last\_len = gost.split\_string\_into\_blocks(bin\_text, 256)

hashed = hex(gost.hash\_gost(blocks, gost.CHANGE\_TABLE, last\_len))

return hashed

def verify(public\_key, signature, hash):

return decrypt(public\_key, signature) == hash

main.py

import digital\_signature

p = 17

q = 19

public\_key, private\_key = digital\_signature.generate\_key\_pair(p, q)

print('Open Key (Public Key): ', public\_key)

print('Private Key: ', private\_key)

with open('text.txt', 'r') as file:

source\_text = file.read()

hash\_value = digital\_signature.hash\_function(source\_text)

print('Hash of the source text (before signing):', hash\_value)

encrypted\_hash = digital\_signature.encrypt(private\_key, hash\_value)

print('Digital Signature (Encrypted hash):', encrypted\_hash)

decrypted\_hash = digital\_signature.decrypt(public\_key, encrypted\_hash)

print('Decrypted hash:', decrypted\_hash)

if digital\_signature.verify(public\_key, encrypted\_hash, hash\_value):

print('Signature verification result: Accept (signature is valid)')

else:

print('Signature verification result: Not accept (signature is invalid)')