Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №6
на тему

ЦИФРОВАЯ ПОДПИСЬ

Выполнил: студент гр. 253503
Минич С.В.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В качестве объекта для практического изучения методов электронной цифровой подписи (ЭЦП) был выбран алгоритм, основанный на отечественном стандарте ГОСТ 34.10. Этот алгоритм обеспечивает гарантированную аутентичность и целостность электронных документов за счет преобразования цифрового сообщения в уникальную подпись, которая используется для подтверждения авторства и защиты информации от несанкционированных изменений.

Алгоритм ГОСТ 34.10, разработанный на базе отечественного стандарта, применяет принципы асимметричной криптографии на основе эллиптических кривых и использует специальные параметры и математические операции сложения и умножения точек для повышения криптостойкости. Непосредственно для создания цифрового сообщения в рамках ЭЦП используется хэш-функция ГОСТ 34.11, которая применяется в протоколах передачи данных и является обязательной частью процесса формирования подписи. Криптостойкость алгоритма ГОСТ 34.10 обеспечивается сложностью решения задачи на эллиптических кривых, что делает его устойчивым к вычислительным атакам [1].

1 ХОД ВЫПОЛНЕНИЯ РАБОТЫ

В ходе лабораторной работы была реализована проверка работы алгоритма ЭЦП ГОСТ 34.10 на языке *Python*. Для обеспечения криптографических операций на эллиптических кривых были реализованы фундаментальные математические функции: вычисление обратного элемента по модулю (*mod_inverse*) с использованием расширенного алгоритма Евклида (*extended_gcd*), а также операции сложения точек (*point_add*) и умножения точки на скаляр (*point_multiply*), которое является ключевым для криптографии на эллиптических кривых.

Для реализации алгоритма формирования подписи была использована следующая последовательность шагов алгоритма хем-функции на базе ГОСТ 34.11.

Кроме того, в работе была реализована процедура проверки подписи. Она включала вычисление вспомогательных значений, а также хэша сообщения.

В практической части лабораторной работы проверялась корректность верификации подписи для тестового сообщения. Полученные результаты подтвердили успешную реализацию алгоритма ГОСТ 34.10, продемонстрировав, что при корректной подписи совпадение достигается. Вывод результата представлен на рисунке 1.

Рисунок 1 – Результат выполнения алгоритма

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы был подробно изучен и реализован процесс формирования и проверки электронной цифровой подписи (ЭЦП) на примере алгоритма ГОСТ 34.10. Была создана программа на *Python*, включающая основные математические операции: вычисление обратного элемента по модулю, сложение точек и скалярное умножение точки, а также шаги формирования подписи и процедуру ее верификации.

Реализация позволила продемонстрировать работу ключевых элементов алгоритма ГОСТ 34.10: применение хэш-функции ГОСТ 34.11 для получения цифрового отпечатка, а также корректность проверки путем сравнения восстановленного значения с исходным.

Цель работы была достигнута: изучение алгоритма ГОСТ 34.10 и его программная реализация позволили на практике закрепить понимание принципов работы асимметричной криптографии на эллиптических кривых, методов защиты информации.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Криптографическая защита информации. [Электронный ресурс]. – Режим доступа: https://meganorm.ru/Data2/1/4293732/4293732954.pdf – Дата доступа: 16.11.2025

ЛИСТИНГ ПРОГРАММНОГО КОДА

```
import secrets
from typing import Tuple
from l5 import hash_gost

def hash(data: bytes, digest_size: int = 256):
    if digest_size == 256:
        return hash_gost(data, 256)
    else:
        return hash_gost(data, 512)

def mod_inverse(a: int, m: int) -> int:
    if a < 0:
        a = (a % m + m) % m
    g, x, _ = extended_gcd(a, m)
    if g != 1:
        raise ValueError("Обратного элемента не существует")
    return x % m

def extended_gcd(a: int, b: int) -> Tuple[int, int, int]:
    if a == 0:
        return b, 0, 1
    gcd, x1, y1 = extended_gcd(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y

def point_add(x1: int, y1: int, x2: int, y2: int):
    if x1 == x2 and y1 == y2:
        lambda_val = (3 * x1 * x1 + a) * mod_inverse(2 * y1, p) % p
    else:
        lambda_val = ((y2 - y1) * mod_inverse(x2 - x1, p)) % p

    x3 = (lambda_val * lambda_val - x1 - x2) % p
    y3 = (lambda_val * (x1 - x3) - y1) % p
    return x3, y3

def point_multiply(k: int, x: int, y: int):
    result_x, result_y = None, None
    buf_x, buf_y = x, y

    while k:
        if k & 1:
            if result_x is None:
                result_x, result_y = buf_x, buf_y
            else:
                result_x, result_y = point_add(result_x, result_y, buf_x, buf_y)

        buf_x, buf_y = point_add(buf_x, buf_y, buf_x, buf_y)
        k >>= 1

    return result_x, result_y

p = 0x8000000000000000000000000000000000000000000000000000000000000431
a = 0x7
b = 0x5FBFF498AA938CE739B8E022FBAFEF40563F6E6A3472FC2A514C0CE9DAE23B7E
m = 0x8000000000000000000000000000000150FE8A1892976154C59CFC193ACCF5B3
q = 0x8000000000000000000000000000000150FE8A1892976154C59CFC193ACCF5B3
xp = 0x2
yp = 0x8E2A8A0E65147D4BD6316030E16D19C85C97F0A9CA267122B96ABBCEA7E8FC8
d = 0x7A929ADE789BB9BE10ED359DD39A72C11B60961F49397EEE1D19CE9891EC3B28
xq, yq = point_multiply(d, xp, yp)

if __name__ == "__main__":
    msg = b"Hello, world!"
    alpha = hash(msg, 256)
    e = alpha % q
    if e == 0:
        e = 1
    r = 0
    s = 0
    while r == 0 or s == 0:
        k = secrets.randbelow(q - 1) + 1
        xc, yc = point_multiply(k, xp, yp)
        r = xc % q
        s = (r*d + k*e) % q

    r_bytes = r.to_bytes(32)
    s_bytes = s.to_bytes(32)
    signature = r_bytes + s_bytes

    print("Подпись:",hex(int.from_bytes(signature)))

    if len(signature) != 64:
        raise ValueError("Подпись неверна")

    extracted_r = int.from_bytes(signature[:32])
    extracted_s = int.from_bytes(signature[32:])
```

```
if extracted_r == 0 or extracted_s == 0:
    raise ValueError("Подпись неверна")

if extracted_r > q or extracted_s > q:
    raise ValueError("Подпись неверна")

extracted_alpha = hash(msg, 256)
extracted_e = extracted_alpha % q
if extracted_e == 0:
    extracted_e = 1

extracted_v = mod_inverse(extracted_e, q)
extracted_z1 = (extracted_s * extracted_v) % q
extracted_z2 = ((-1) * extracted_r * extracted_v) % q

first_x, first_y = point_multiply(extracted_z1, xp, yp)
second_x, second_y = point_multiply(extracted_z2, xq, yq)
extracted_xc, extracted_yc = point_add(first_x, first_y, second_x, second_y)
extracted_R = extracted_xc % q

if extracted_R != extracted_r:
    print("Подпись неверна")
else:
    print("Подпись верна")
```