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 №7
на тему

КРИПТОГРАФИЯ С ИСПОЛЬЗОВАНИЕМ ЭЛЛИПТИЧЕСКИХ КРИВЫХ

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 криптографии с использованием эллиптических кривых был выбран алгоритм шифрования, представляющий собой аналог классической схемы Эль-Гамаля, адаптированный для работы в группе точек эллиптической кривой. Данный подход позволяет обеспечить конфиденциальность передаваемых данных за счёт использования механизма шифрования и дешифрования, при котором зашифрование выполняется с помощью открытого ключа получателя, а расшифрование только с использованием соответствующего ему закрытого ключа.

Алгоритм Эль-Гамаля на эллиптических кривых базируется на фундаментальных принципах асимметричной криптографии. В рамках схемы выбирается эллиптическая кривая с заданными параметрами (a, b, p, G, n), где G – базовая точка. Процесс шифрования заключается в представлении открытого текста в виде точки на кривой, передачи пары точек шифротекста. Дешифрование осуществляется путём вычисления общей секретной точки с использованием закрытого ключа получателя и последующего восстановления исходного сообщения.

В ходе выполнения лабораторной работы будет реализована схема шифрования и дешифрования на основе эллиптической кривой, включая генерацию ключевой пары, преобразование сообщения в точку кривой при помощи метода Коблица, формирование шифротекста и восстановление открытого текста.

1 ХОД ВЫПОЛНЕНИЯ РАБОТЫ

В ходе лабораторной работы была реализована схема асимметричного шифрования и дешифрования сообщений, представляющая собой аналог алгоритма Эль-Гамаля, адаптированный для группы точек эллиптической кривой. Реализация выполнена на языке программирования *Python*.

Особое внимание уделено процедуре обратимого отображения произвольного текстового сообщения в точку эллиптической кривой, методу Коблица. Для этого исходное сообщение разбивается на блоки фиксированной длины, каждый блок преобразуется в большое целое число m, после чего формируется кандидат на координату x путём присоединения к m младших битов-параметров j. Далее вычисляется значение $y^2 = x^3 + ax + b \ (mod\ p)$. Если полученное значение является квадратичным вычетом по модулю p, вычисляется квадратный корень y по модулю p. В случае неудачи параметр j увеличивается на единицу, и проверка повторяется. Благодаря выбранному размеру блока и количеству попыток вероятность неудачи при кодировании одного блока пренебрежимо мала.

Непосредственно процесс шифрования заключается в выборе случайного ключа k, вычислении точки $C_1 = k \cdot G$. Точка-сообщение M маскируется путём сложения с секретной точкой: $C_2 = M + k \cdot Q$. Шифротекстом для каждого блока является пара точек (C_1, C_2). Дешифрование выполняется вычислением той же секретной точки как $d \cdot C_1$ с последующим вычитанием её из C_2 и обратным декодированием полученной точки M в исходное сообщение.

Полученные результаты подтвердили корректность реализации всех этапов алгоритма, включая операции над эллиптической кривой и обратимое отображение сообщения в точку по методу Коблица. Вывод результатов выполнения программы представлен на рисунке 1.

```
PS C:\sem7\MZI> & C:/Users/imsve/AppData/Local/Programs/Python/Python312/python.exe c:/sem7/MZI/lab7/l7_b
ig.py
Исходное сообщение: b' '
Расшифрованное сообщение: b' '
Совпадают: True

Исходное сообщение: b'Hello, world!'
Расшифрованное сообщение: b'Hello, world!'
Совпадают: True

Исходное большое сообщение: b'Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello
, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world
! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! '
Расшифрованное сообщение: b'Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello,
world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world!
Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! Hello, world! '
Совпадают: True
```

Рисунок 1 – Результат выполнения алгоритма

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы был подробно изучен и практически реализован алгоритм асимметричного шифрования, представляющий собой аналог схемы Эль-Гамаля на эллиптических кривых. На языке программирования *Python* создана программа, включающая все необходимые криптографические примитивы: арифметику эллиптической кривой (сложение и умножение точек), обратимое отображение произвольных байтовых сообщений в точки кривой по методу Коблица, а также непосредственно процедуры шифрования и дешифрования сообщений произвольной длины.

Особое внимание уделено надёжной и эффективной реализации кодирования/декодирования сообщений в точки эллиптической кривой. Проведённое тестирование на сообщениях малой и большой длины подтвердило полную корректность всех этапов: после шифрования открытым ключом и последующего дешифрования соответствующим закрытым ключом исходное сообщение восстанавливалось без единого искажения.

Цель лабораторной работы достигнута: выполнена программная реализация и практическая проверка одной из базовых схем асимметричного шифрования на эллиптических кривых.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] НОУ ИНТУИТ | Математика криптографии и теория шифрования. Лекция 15: Криптосистемы. [Электронный ресурс]. – Режим доступа: https://intuit.ru/studies/courses/552/408/lecture/9373?page=6– Дата доступа: 29.11.2025

ЛИСТИНГ ПРОГРАММНОГО КОДА

```
import random

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

def extended_gcd(a: int, b: 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, a: int, p: int):
    if x1 == x2 and y1 == y2:
        lambda_val = (3 * x1 * x1 + a) * mod_inverse(2 * y1, p) % p
    elif x1 == x2:
        return None, None
    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, a: int, p: 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, a, p)

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

    return result_x, result_y

def sqrt_mod_p(a: int, p: int):
    a = a % p
    if a == 0:
        return 0
    if pow(a, (p - 1) // 2, p) != 1:
        return None
    return pow(a, (p + 1) // 4, p)

def encode_message(data: bytes, a: int, b: int, p: int, k_bits: int = 40):
    m = int.from_bytes(data, 'big')
    for j in range(1 << k_bits):
        x = (m << k_bits) | j
        x %= p
        y2 = (pow(x, 3, p) + a * x + b) % p
        y = sqrt_mod_p(y2, p)
        if y is not None:
            return x, y
    raise Exception("Не удалось закодировать сообщение в точку на кривой")

def decode_message(x: int, y: int, k_bits: int = 40):
    m = x >> k_bits
    byte_len = (m.bit_length() + 7) // 8
    if byte_len == 0:
        return b'\x00'
    return m.to_bytes(byte_len, 'big')

def encrypt_point(msg_x: int, msg_y: int, pub_x: int, pub_y: int, gx: int, gy: int, a: int, p: int, n:
int):
    k = random.randint(1, n - 1)
    c1_x, c1_y = point_multiply(k, gx, gy, a, p)
    shared_x, shared_y = point_multiply(k, pub_x, pub_y, a, p)
    c2_x, c2_y = point_add(msg_x, msg_y, shared_x, shared_y, a, p)
    return (c1_x, c1_y), (c2_x, c2_y)

def decrypt_point(c1: tuple, c2: tuple, d: int, a: int, p: int):
    shared_x, shared_y = point_multiply(d, c1[0], c1[1], a, p)
    msg_x, msg_y = point_add(c2[0], c2[1], shared_x, (-shared_y) % p, a, p)
    return msg_x, msg_y

def encrypt_data(data: bytes, pub_x: int, pub_y: int, gx: int, gy: int, a: int, b: int, p: int, n: int,
k_bits: int = 40):
```

```
    max_chunk_size = (p.bit_length() - k_bits - 8) // 8
    chunks = [data[i:i + max_chunk_size] for i in range(0, len(data), max_chunk_size)]

    encrypted_chunks = []
    for chunk in chunks:
        if not chunk:
            continue
        msg_x, msg_y = encode_message(chunk, a, b, p, k_bits)
        c1, c2 = encrypt_point(msg_x, msg_y, pub_x, pub_y, gx, gy, a, p, n)
        encrypted_chunks.append((c1, c2))

    return encrypted_chunks

def decrypt_data(encrypted_chunks: list, d: int, a: int, p: int, k_bits: int = 40):
    decrypted_data = b''
    for c1, c2 in encrypted_chunks:
        msg_x, msg_y = decrypt_point(c1, c2, d, a, p)
        chunk = decode_message(msg_x, msg_y, k_bits)
        decrypted_data += chunk
    return decrypted_data


P = 0x8000000000000000000000000000000000000000000000000000000000000431
A = 0x7
B = 0x5FBFF498AA938CE739B8E022FBAFEF40563F6E6A3472FC2A514C0CE9DAE23B7E
N = 0x800000000000000000000000000000150FE8A1892976154C59CFC193ACCF5B3
GX = 0x2
GY = 0x8E2A8A0E65147D4BD6316030E16D19C85C97F0A9CA267122B96ABBCEA7E8FC8
K_BITS = 40

# https://intuit.ru/studies/courses/552/408/lecture/9373?page=6
if __name__ == "__main__":
    d = random.randint(1, N - 1)
    pub_x, pub_y = point_multiply(d, GX, GY, A, P)

    msg = b" "
    print(f"Исходное сообщение: {msg}")

    encrypted = encrypt_data(msg, pub_x, pub_y, GX, GY, A, B, P, N, K_BITS)
    decrypted = decrypt_data(encrypted, d, A, P, K_BITS)
    print(f"Расшифрованное сообщение: {decrypted}")
    print(f"Совпадают: {msg == decrypted}")

    msg = b"Hello, world!"
    print(f"\nИсходное сообщение: {msg}")

    encrypted = encrypt_data(msg, pub_x, pub_y, GX, GY, A, B, P, N, K_BITS)
    decrypted = decrypt_data(encrypted, d, A, P, K_BITS)
    print(f"Расшифрованное сообщение: {decrypted}")
    print(f"Совпадают: {msg == decrypted}")

    large_msg = b"Hello, world! " * 20
    print(f"\nИсходное большое сообщение: {large_msg}")

    encrypted_large = encrypt_data(large_msg, pub_x, pub_y, GX, GY, A, B, P, N, K_BITS)
    decrypted_large = decrypt_data(encrypted_large, d, A, P, K_BITS)
    print(f"Расшифрованное сообщение: {decrypted_large}")
    print(f"Совпадают: {large_msg == decrypted_large}")
```