

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

**АЛГОРИТМА ЭЛЬ-ГАМАЛЯ НА ОСНОВЕ ЭЛЛИПТИЧЕСКИХ
КРИВЫХ**

Выполнил: студент гр.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики

Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 Цель работы	3
2 Этапы выполнения работы.....	4
2.1 Краткие теоретические сведения	4
2.2 Пример работы программы.....	4
Заключение	6
Приложение А (обязательное) Листинг программного кода	7

1 ЦЕЛЬ РАБОТЫ

Целью данной лабораторной работы является практическое изучение и реализация асимметричной криптографической системы Эль-Гамала на основе эллиптических кривых, которая представляет собой одну из современных криптосистем с открытым ключом. Работа направлена на глубокое понимание математических основ эллиптической криптографии, в частности операций с точками на кривой и принципов скалярного умножения.

В рамках лабораторной работы предстоит разработать программное обеспечение на языке *Python*, реализующее полный цикл работы системы Эль-Гамала: от генерации ключевой пары до шифрования и дешифрования текстовых и бинарных данных. Особое внимание уделяется кодированию блоков данных в точки кривой, использованию случайного скаляра для формирования шифртекста и процедурам восстановления исходных данных при помощи приватного ключа.

Важной составляющей работы является исследование особенностей схемы Эль-Гамала, связанных с обработкой блоков и проверкой их целостности с помощью *CRC*. Практическая задача включает разработку механизма надежного кодирования и декодирования данных, сжатия и восстановления точек кривой, а также контроля корректности расшифровки.

Результатом работы должно стать законченное программное решение, сопровождаемое тестовыми примерами, демонстрирующими корректность работы всех компонентов системы: генерации ключей, шифрования и дешифрования. Полученные знания и навыки имеют важное значение для понимания принципов построения современных криптографических систем на эллиптических кривых и могут быть применены в дальнейшем при изучении более сложных криптографических протоколов и алгоритмов.

2 ЭТАПЫ ВЫПОЛНЕНИЯ РАБОТЫ

2.1 Краткие теоретические сведения

Криптосистема Эль-Гамала на эллиптических кривых, впервые предложенная в 1985 году, представляет собой асимметричную криптографическую схему, безопасность которой напрямую связана с вычислительной сложностью задачи дискретного логарифмирования на эллиптической кривой. В отличие от многих других асимметричных алгоритмов, для схемы Эль-Гамала существует строгое математическое обоснование её стойкости: успешный криптоанализ эквивалентен нахождению скаляра k по известным точкам G и $Q = kG$, что при корректно выбранных параметрах кривой остаётся вычислительно трудной задачей.

Основу алгоритма составляет трудность обращения операции скалярного умножения точки на кривой: для публичной точки $Q = dG$ и базовой точки G невозможно эффективно определить приватный ключ d . Эта асимметрия обеспечивает надёжность шифрования и является фундаментальной характеристикой системы.

Шифрование в схеме Эль-Гамала на эллиптических кривых заключается в генерации случайного скаляра k и вычислении пары точек ($C1 = kG, C2 = M + kQ$), где M — точка, закодированная из блока данных. Эта операция выполняется эффективно даже для больших параметров кривой. Однако обратная операция — восстановление исходной точки M без знания приватного ключа d — является вычислительно трудной задачей, обеспечивая криптографическую стойкость схемы.

Одной из особенностей реализации Эль-Гамала является необходимость кодирования данных в точки кривой и контроля их целостности с помощью CRC. Это позволяет корректно восстанавливать исходные сообщения и предотвращает ошибки при дешифровании. Для практических целей используется сжатие точек и проверка блоков данных, что обеспечивает надёжность и эффективность алгоритма при работе с произвольными текстовыми и бинарными данными.

2.2 Пример работы программы

На вход программы подается текстовый файл с исходным текстом или бинарными данными для шифрования. На рисунке 2.2.1 представлено содержимое исходного файла перед шифрованием. Программа выполняет генерацию ключевой пары, включающей приватный ключ d и соответствующий публичный ключ $Q = dG$, где G — базовая точка эллиптической кривой.

На рисунке 2.2.2 демонстрируется процесс генерации ключей и основные параметры криптосистемы, включая координаты базовой точки, приватный ключ и публичный ключ. Особенностью схемы Эль-Гамала является необходимость кодирования данных в точки кривой, что показано на

рисунке 2.2.3: каждый блок данных преобразуется в точку M , после чего формируется пара шифротекста ($C1, C2$).

Программа обеспечивает корректное восстановление исходного сообщения на этапе дешифрования с помощью приватного ключа d , проверки CRC и контроля целостности данных. На рисунке 2.2.4 показан окончательный результат работы программы — успешно расшифрованный текст, полностью соответствующий исходному сообщению.

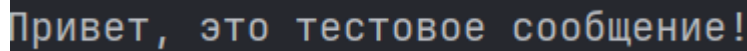


Рисунок 2.2.1 – Исходный текст

```
2025-09-25 13:27:32 | INFO | Сгенерированы ключи.
2025-09-25 13:27:32 | INFO |   priv: 3d681b60cd6081797c389dfc6063cd6be64f7ba07910691771878181136e1217
2025-09-25 13:27:32 | INFO |   pub : 02216b5cc73106ed8f21318c7b15c4d46a127cef72e335c0e93a2956b458ccd7a8
```

Рисунок 2.2.2 – Демонстрация генерации ключей

```
2025-09-25 13:27:44 | INFO | Ключи загружены. priv=3d681b60cd608179...71878181136e1217..., pub=02216b5cc73106ed...3a2956b458ccd7a8...
2025-09-25 13:27:44 | INFO | Создан тестовый input.txt
2025-09-25 13:27:44 | INFO | Читаем input.txt: 57 байт
2025-09-25 13:27:44 | INFO | Этап шифрования: блоков 3 (данных 30 байт + CRC1 = 31 байт на блок)
2025-09-25 13:27:44 | INFO | Блок 1/3: CRC=24, m=39d09fd180d0b8d0...be20d182d0b5d124 -> C1=023199dc2d1e7d41...944bb34f02d07337 ;
00c0b4b482d0c9a
2025-09-25 13:27:44 | INFO | Блок 2/3: CRC=44, m=81d182d0bed0b2d0...b5d0bdd0b8d0b544 -> C1=039af0d8a5de8e47...acceeb0363262263f ;
deae339e166c28c
2025-09-25 13:27:44 | INFO | Блок 3/3: CRC=22, m=2100000000000000...0000000000000022 -> C1=0314bf4b5be1540e...e86e5ebc3dbd1a41 ;
429989fd4481a41
2025-09-25 13:27:44 | INFO | Шифротекст записан в cipher.txt: 3 строки
OK: input.txt -> cipher.txt
```

Рисунок 2.2.3 – Шифрование исходного текста

```
2025-09-25 13:27:53 | INFO | Загружен приватный ключ: 3d681b60cd608179...71878181136e1217...
2025-09-25 13:27:53 | INFO | Читаем cipher.txt: 3 строки
2025-09-25 13:27:53 | INFO | Этап дешифрования: пар 3
2025-09-25 13:27:53 | INFO | Пара 1/3: C1=023199dc2d1e7d41...944bb34f02d07337 ; C2=0317f1e38556bfa7...900c0b4b482d0c9a -> CRC=24 OK
2025-09-25 13:27:53 | INFO | Пара 2/3: C1=039af0d8a5de8e47...acceeb0363262263f ; C2=028d4583aa52a780...8deae339e166c28c -> CRC=44 OK
2025-09-25 13:27:53 | INFO | Пара 3/3: C1=0314bf4b5be1540e...e86e5ebc3dbd1a41 ; C2=027273461236b2b1...7429989fd4481a41 -> CRC=22 OK
2025-09-25 13:27:53 | INFO | Готово: полезная длина 57 байт
2025-09-25 13:27:53 | INFO | Расшифрованный UTF-8 записан в output.txt
2025-09-25 13:27:53 | INFO | Превью: "Привет, это тестовое сообщение!"
OK: cipher.txt -> output.txt
```

Рисунок 2.2.4 – Расшифрованный текст

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была успешно реализована и протестирована криптосистема Эль-Гамала на эллиптических кривых — одна из современных асимметричных криптографических схем. Практическая реализация позволила детально изучить её математическую основу, связанную с вычислительной сложностью задачи дискретного логарифмирования на эллиптической кривой и операциями скалярного умножения точек.

В процессе работы были отработаны ключевые компоненты алгоритма: генерация приватного ключа d и соответствующего публичного ключа $Q = dG$, кодирование блоков данных в точки кривой, шифрование с использованием случайного скаляра k и формирование пар точек $(C1, C2)$, а также процедура дешифрования с проверкой CRC и восстановления исходного сообщения. Для обеспечения корректного восстановления данных был реализован механизм контроля целостности и семантической корректности расшифрованных блоков.

Практическая ценность выполненной работы заключается в создании полнофункционального программного инструмента, способного корректно выполнять шифрование и дешифрование текстовой и бинарной информации. При этом были приобретены важные навыки работы с арифметикой точек на эллиптической кривой, обработкой блоков данных и реализацией криптографических примитивов — компетенции, имеющие значительное значение в сфере информационной безопасности.

Экспериментальные результаты полностью подтвердили теоретические свойства схемы: преобразования оказались обратимыми при условии знания приватного ключа, а криптостойкость обусловлена трудностью вычисления дискретного логарифма на эллиптической кривой. Разработанное решение может эффективно использоваться в учебных целях для демонстрации принципов асимметричной криптографии и практического применения методов эллиптической криптографии в задачах защиты информации.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

```
import logging
import os
import secrets
from typing import Optional, Tuple, List

PRIV_PATH = "priv.hex"
PUB_PATH = "pub.hex"
PLAIN_IN = "input.txt"
CIPHER_IO = "cipher.txt"
PLAIN_OUT = "output.txt"

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s | %(levelname)s | %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)
log = logging.getLogger("ECC-ElGamal")

p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEC2F
a = 0
b = 7
Gx = 55066263022277343669578718895168534326250603453777594175500187360389116729240
Gy = 32670510020758816978083085130507043184471273380659243275938904335757337482424
n = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141

Point = Optional[Tuple[int, int]]
G: Point = (Gx, Gy)
def short_hex(b: bytes, take: int = 8) -> str:
    h = b.hex()
    return h if len(h) <= 2 * take else f"{h[:2*take]}...{h[-2*take:]}"
def short_int(x: int, take: int = 8) -> str:
    h = f"{x:0x}"
    return h if len(h) <= 2 * take else f"{h[:2*take]}...{h[-2*take:]}"
def is_on_curve(P: Point) -> bool:
    if P is None:
        return True
    x, y = P
    return (y * y - (x * x * x + a * x + b)) % p == 0
def mod_inv(x: int, m: int) -> int:
    return pow(x, -1, m)
def point_add(P: Point, Q: Point) -> Point:
    if P is None:
        return Q
    if Q is None:
        return P
    x1, y1 = P
    x2, y2 = Q
    if x1 == x2 and (y1 + y2) % p == 0:
        return None
    if P == Q:
        if y1 == 0:
            return None
        s = (3 * x1 * x1 + a) * mod_inv(2 * y1 % p, p) % p
    else:
```

```

        if x1 == x2:
            return None
        s = (y2 - y1) * mod_inv((x2 - x1) % p, p) % p
        x3 = (s * s - x1 - x2) % p
        y3 = (s * (x1 - x3) - y1) % p
        return (x3, y3)

def point_neg(P: Point) -> Point:
    if P is None:
        return None
    x, y = P
    return (x, (-y) % p)

def point_mul(P: Point, k: int) -> Point:
    assert is_on_curve(P)
    if k % n == 0 or P is None:
        return None
    k %= n
    R = None
    Q = P
    while k:
        if k & 1:
            R = point_add(R, Q)
            Q = point_add(Q, Q)
            k >>= 1
    assert is_on_curve(R)
    return R

assert is_on_curve(G)

def mod_sqrt(a_: int) -> Optional[int]:
    a_ %= p
    if a_ == 0:
        return 0
    y = pow(a_, (p + 1) // 4, p)
    return y if (y * y - a_) % p == 0 else None

T = 256
BLOCK_DATA_BYTES = 30
BLOCK_TOTAL_BYTES = 31

def crc8(bs: bytes) -> int:
    crc = 0
    for b in bs:
        crc ^= b
        for _ in range(8):
            crc = ((crc << 1) ^ 0x07) & 0xFF if (crc & 0x80) else (crc << 1) &
0xFF
    return crc

def bytes_to_int(b: bytes) -> int:
    return int.from_bytes(b, "big")

def int_to_bytes(x: int, length: int) -> bytes:
    return x.to_bytes(length, "big")

```



```

def encode_block_to_point(m: int) -> Point:
    assert 0 <= m < (p - 1) // T
    base = m * T
    j_max = min(T - 1, (p - 1) - base)
    for j in range(j_max + 1):
        x = base + j
        rhs = (pow(x, 3, p) + a * x + b) % p
        y = mod_sqrt(rhs)
        if y is not None:
            if y & 1:
                y = (-y) % p
            P = (x, y)
            assert is_on_curve(P)
            return P
    raise ValueError(
        "Не удалось закодировать блок в точку (уменьшите BLOCK_DATA_BYTES или
увеличьте T)."
    )

def decode_point_to_block(P: Point) -> int:
    if P is None:
        raise ValueError("∞ не кодирует блок")
    x, _ = P
    return x // T

def chunk_bytes(data: bytes, size: int) -> List[bytes]:
    return [data[i : i + size] for i in range(0, len(data), size)]

def compress(P: Point) -> bytes:
    if P is None:
        return b"\x00"
    x, y = P
    prefix = 0x02 if (y % 2 == 0) else 0x03
    return bytes([prefix]) + x.to_bytes(32, "big")

def decompress(bts: bytes) -> Point:
    if bts == b"\x00":
        return None
    if len(bts) != 33 or bts[0] not in (2, 3):
        raise ValueError(
            "Неверный формат сжатой точки (ожидается 33 байта, префикс
0x02/0x03)"
        )
    prefix = bts[0]
    x = int.from_bytes(bts[1:], "big")
    rhs = (pow(x, 3, p) + a * x + b) % p
    y = mod_sqrt(rhs)
    if y is None:
        raise ValueError(
            "Не удаётся восстановить Y: точка не на кривой (возможна порча
строки)."
        )
    if (y % 2 == 0 and prefix == 3) or (y % 2 == 1 and prefix == 2):
        y = (-y) % p
    P = (x, y)
    assert is_on_curve(P)
    return P

```

```

def generate_keypair() -> Tuple[int, Point]:
    while True:
        d = secrets.randbelow(n)
        if 1 <= d < n:
            Q = point_mul(G, d)
            if Q is not None:
                return d, Q

def encrypt_point(M: Point, Qpub: Point) -> Tuple[Point, Point]:
    if not is_on_curve(Qpub) or Qpub is None:
        raise ValueError("Публичный ключ некорректен")
    k = secrets.randbelow(n - 1) + 1
    C1 = point_mul(G, k)
    kQ = point_mul(Qpub, k)
    C2 = point_add(M, kQ)
    return C1, C2

def decrypt_point(C1: Point, C2: Point, d: int) -> Point:
    dC1 = point_mul(C1, d)
    return point_add(C2, point_neg(dC1))

def pairs_to_lines(pairs: List[Tuple[bytes, bytes]]) -> List[str]:
    return [f"{c1.hex()};{c2.hex()}" for c1, c2 in pairs]

def lines_to_pairs(lines: List[str]) -> List[Tuple[bytes, bytes]]:
    out: List[Tuple[bytes, bytes]] = []
    for idx, ln in enumerate(lines, 1):
        ln = ln.strip()
        if not ln:
            log.info("Строка %d пустая — пропущена", idx)
            continue
        if ";" not in ln:
            raise ValueError(f"Строка {idx}: отсутствует ';' между C1 и C2.")
        clh, c2h = ln.split(";", 1)
        try:
            clb = bytes.fromhex(clh)
            c2b = bytes.fromhex(c2h)
        except ValueError as e:
            raise ValueError(f"Строка {idx}: невалидный hex: {e}")
        out.append((clb, c2b))
    return out

def encrypt_bytes(plain: bytes, Qpub: Point) -> List[Tuple[bytes, bytes]]:
    payload = len(plain).to_bytes(4, "big") + plain

    blocks = chunk_bytes(payload, BLOCK_DATA_BYTES)
    log.info(
        "Этап шифрования: блоков %d (данных %d байт + CRC1 = %d байт на блок)",
        len(blocks),
        BLOCK_DATA_BYTES,
        BLOCK_TOTAL_BYTES,
    )

    pairs: List[Tuple[bytes, bytes]] = []
    for i, data_block in enumerate(blocks, 1):
        if len(data_block) < BLOCK_DATA_BYTES:
            data_block = data_block + b"\x00" * (BLOCK_DATA_BYTES -
len(data_block))
            c = crc8(data_block)

```

```

block_with_crc = data_block + bytes([c])

m = bytes_to_int(block_with_crc)
assert m < (p - 1) // T, "m слишком велик, уменьшите BLOCK_DATA_BYTES"

Pm = encode_block_to_point(m)
C1, C2 = encrypt_point(Pm, Qpub)
pairs.append((compress(C1), compress(C2)))

if i <= 3 or i == len(blocks):
    log.info(
        " Блок %d/%d: CRC=%02x, m=%s -> C1=%s ; C2=%s",
        i,
        len(blocks),
        c,
        short_int(m),
        short_hex(compress(C1)),
        short_hex(compress(C2)),
    )
elif i % 100 == 0:
    log.info(" ... обработано %d блоков", i)
return pairs

def decrypt_bytes(pairs: List[Tuple[bytes, bytes]], d: int) -> bytes:
    buf = bytearray()
    total = len(pairs)
    log.info("Этап дешифрования: пар %d", total)
    for i, (c1b, c2b) in enumerate(pairs, 1):
        try:
            C1 = decompress(c1b)
            C2 = decompress(c2b)
        except Exception as e:
            raise ValueError(f"Пара {i}: ошибка декомпрессии точек: {e}")
        if not is_on_curve(C1) or not is_on_curve(C2):
            raise ValueError(
                f"Пара {i}: точка(и) не на кривой (повреждение шифртекста)."
            )

        Pm = decrypt_point(C1, C2, d)
        if Pm is None or not is_on_curve(Pm):
            raise ValueError(f"Пара {i}: восстановленная точка сообщения некорректна.")

        m = decode_point_to_block(Pm)
        block_with_crc = int_to_bytes(m, BLOCK_TOTAL_BYTES)
        data_block = block_with_crc[:-1]
        c_stored = block_with_crc[-1]
        c_actual = crc8(data_block)
        if c_actual != c_stored:
            raise ValueError(
                f"Пара {i}: CRC mismatch (ожидалось {c_stored:02x}, вычислено {c_actual:02x})."
            )

        buf.extend(data_block)

    if i <= 3 or i == total:
        log.info(
            " Пара %d/%d: C1=%s ; C2=%s -> CRC=%02x OK",
            i,
            total,
            short_hex(c1b),
            short_hex(c2b),

```

```

        c_stored,
    )
    elif i % 100 == 0:
        log.info(" ... обработано %d пар", i)

    if len(buf) < 4:
        raise ValueError("Данные повреждены: нет 4-байтового префикса длины.")
    L = int.from_bytes(buf[:4], "big")
    data = bytes(buf[4 : 4 + L])
    if len(data) != L:
        raise ValueError(
            f"Некорректная длина при восстановлении: ожидалось {L}, получили {len(data)}."
        )
    log.info("Готово: полезная длина %d байт", L)
    return data

def save_priv_hex(path: str, d: int):
    with open(path, "w", encoding="utf-8") as f:
        f.write(f"{d:064x}\n")

def save_pub_hex(path: str, Q: Point):
    with open(path, "w", encoding="utf-8") as f:
        f.write(compress(Q).hex() + "\n")

def load_priv_hex(path: str) -> int:
    with open(path, "r", encoding="utf-8") as f:
        s = f.read().strip()
    return int(s, 16)

def load_pub_hex(path: str) -> Point:
    with open(path, "r", encoding="utf-8") as f:
        s = f.read().strip()
    return decompress(bytes.fromhex(s))

def cmd_gen_keys():
    d, Q = generate_keypair()
    save_priv_hex(PRIV_PATH, d)
    save_pub_hex(PUB_PATH, Q)
    log.info("Сгенерированы ключи.")
    log.info("  priv: %s", f"{d:064x}")
    log.info("  pub : %s", compress(Q).hex())
    print("OK: создано priv.hex / pub.hex")

def cmd_encrypt():
    if not os.path.exists(PUB_PATH) or not os.path.exists(PRIV_PATH):
        log.info("Ключи не найдены – генерируем...")
        cmd_gen_keys()
    Q = load_pub_hex(PUB_PATH)
    d = load_priv_hex(PRIV_PATH)
    log.info("Ключи загружены.      priv=%s...,      pub=%s...",      short_int(d),
short_hex(compress(Q)))
    if not os.path.exists(PLAIN_IN):
        with open(PLAIN_IN, "w", encoding="utf-8") as f:
            f.write("Привет, это тестовое сообщение!")
        log.info("Создан тестовый %s", PLAIN_IN)
    with open(PLAIN_IN, "r", encoding="utf-8") as f:
        data = f.read().encode("utf-8")

```

```

log.info("Читаем %s: %d байт", PLAIN_IN, len(data))
pairs = encrypt_bytes(data, Q)
lines = pairs_to_lines(pairs)
with open(CIPHER_IO, "w", encoding="utf-8") as f:
    f.write("\n".join(lines) + "\n")
log.info("Шифртекст записан в %s: %d строк", CIPHER_IO, len(lines))
print(f"OK: {PLAIN_IN} -> {CIPHER_IO}")

def cmd_decrypt():
    if not os.path.exists(PRIV_PATH):
        raise SystemExit(
            "Нет приватного ключа priv.hex. Сначала запусти gen-keys/encrypt."
        )
    d = load_priv_hex(PRIV_PATH)
    log.info("Загружен приватный ключ: %s...", short_int(d))
    if not os.path.exists(CIPHER_IO):
        raise SystemExit(f"Нет {CIPHER_IO}. Сначала запусти encrypt.")
    with open(CIPHER_IO, "r", encoding="utf-8") as f:
        lines = [ln.strip() for ln in f if ln.strip()]
    log.info("Читаем %s: %d строк", CIPHER_IO, len(lines))
    pairs = lines_to_pairs(lines)
    plain = decrypt_bytes(pairs, d)
    try:
        text = plain.decode("utf-8")
        preview = text[:80].replace("\n", "\\n")
        with open(PLAIN_OUT, "w", encoding="utf-8") as f:
            f.write(text)
        log.info("Расшифрованный UTF-8 записан в %s", PLAIN_OUT)
        log.info('Превью: "%s"%s', preview, "..." if len(text) > 80 else "")
    except UnicodeDecodeError:
        with open(PLAIN_OUT, "wb") as f:
            f.write(plain)
        log.info("Расшифрованные байты (не UTF-8) записаны в %s", PLAIN_OUT)
        log.info("Превью HEX: %s", short_hex(plain))
    print(f"OK: {CIPHER_IO} -> {PLAIN_OUT}")

def main():
    import sys

    if len(sys.argv) != 2 or sys.argv[1] not in ("gen-keys", "encrypt",
"decrypt"):
        print("Использование:")
        print("  python main.py gen-keys    создать priv.hex/pub.hex")
        print("  python main.py encrypt     input.txt -> cipher.txt")
        print("  python main.py decrypt     cipher.txt -> output.txt")
        raise SystemExit(1)
    cmd = sys.argv[1]
    if cmd == "gen-keys":
        cmd_gen_keys()
    elif cmd == "encrypt":
        cmd_encrypt()
    elif cmd == "decrypt":
        cmd_decrypt()

if __name__ == "__main__":
    main()

```