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](#_Toc190556772)

[2 Этапы выполнения работы 4](#_Toc190556773)

[2.1 Краткие теоретические сведения 4](#_Toc190556774)

[2.2 Пример работы программы 5](#_Toc190556775)

[Заключение 6](#_Toc190556776)  
[Приложение А (обязательное) Листинг программного кода 7](#_Toc209095982)

# 1 ЦЕЛЬ РАБОТЫ

Целью данной лабораторной работы является практическое изучение и реализация асимметричной криптографической системы Эль-Гамаля 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-Гамаля: 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-Гамаля, 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-Гамаля 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-Гамаля 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-Гамаля на эллиптических кривых заключается в генерации случайного скаляра *k* и вычислении пары точек (*C*1 = *kG*,*C*2 = *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-Гамаля 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 надежность 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-Гамаля 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 (*C*1,*C*2).

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

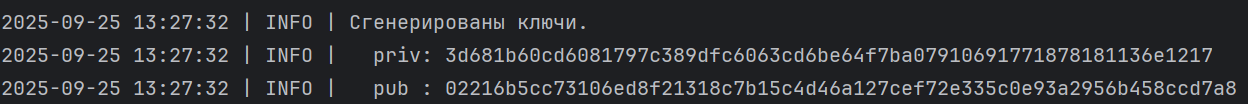


Рисунок 2.2.2 – Демонстрация генерации ключей

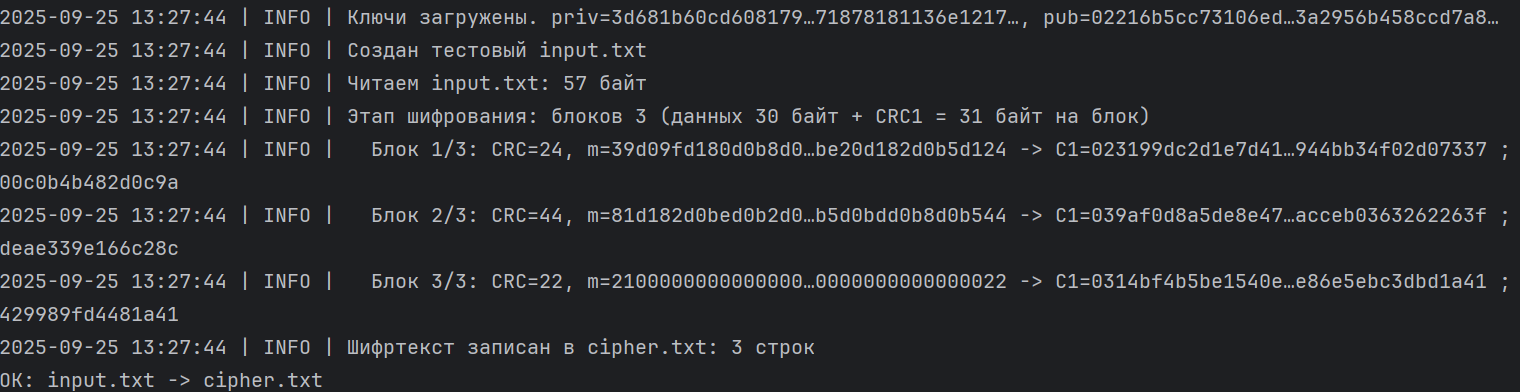


Рисунок 2.2.3 – Шифрование исходного текста

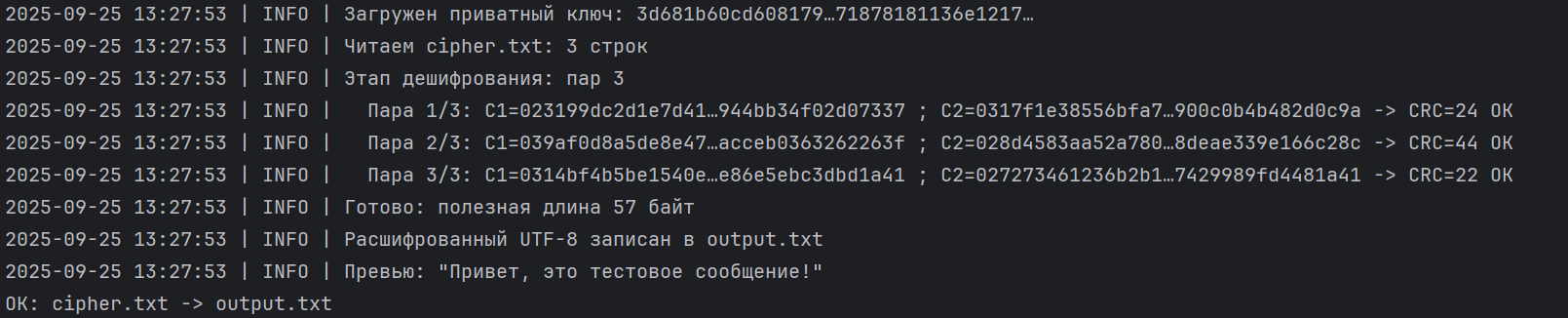


Рисунок 2.2.4 – Расшифрованный текст

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была успешно реализована и протестирована криптосистема Эль-Гамаля 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 (*C*1,*C*2), 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 = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F

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.")

c1h, c2h = ln.split(";", 1)

try:

c1b = bytes.fromhex(c1h)

c2b = bytes.fromhex(c2h)

except ValueError as e:

raise ValueError(f"Строка {idx}: невалидный hex: {e}")

out.append((c1b, 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()