

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 №8  
на тему

**СТЕГАНОГРАФИЧЕСКИЕ МЕТОДЫ**

БГУИР КП 1-40 04 01 025 ПЗ

Выполнил: студент гр.253504  
Фроленко К.Ю.

Проверил: ассистент кафедры информатики  
Герчик А.В.

Минск 2025

## СОДЕРЖАНИЕ

1 Формулировка задачи .....	3
2 Теоретические сведения .....	3
3 Ход работы.....	6
Заключение .....	8
Приложение А (обязательное) Листинг программного кода .....	9

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью данной лабораторной работы является изучение и практическая реализация методов стеганографии в частотной области цифровых изображений. Конкретно требуется:

1 Ознакомиться с теоретическими основами работы в частотной области (*DCT*, блоки  $8 \times 8$ , спектральные коэффициенты), понять отличие частотных методов от методов пространственной области (например, *LSB*);

2 Реализовать программное средство, позволяющее скрывать текстовые сообщения в изображении и извлекать их обратно, работающее через файловый ввод/вывод (вход/выход — *in.txt* / *out.txt*) и имеющее консольный интерфейс;

3 Обеспечить поддержку *Unicode* (*UTF-8*) при кодировании и извлечении сообщений;

4 Провести тестирование реализованного алгоритма и продемонстрировать корректность работы на примере.

Практическая часть направлена на закрепление знаний по обработке цифровых изображений, работе с *DCT*, организации файлового ввода-вывода и обработке побитовых представлений данных.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Стеганография — это наука о сокрытии информации таким образом, чтобы сам факт её передачи оставался незаметным. Если криптография защищает содержимое сообщения, то стеганография прячет сам факт его существования. В цифровых изображениях применяется несколько подходов, среди которых выделяют методы пространственной и частотной областей. Пространственные методы напрямую изменяют значения пикселей, например, заменяют младшие биты каждого канала в соответствии с битами сообщения. Эти методы просты в реализации и позволяют спрятать достаточно много информации, но они плохо защищены от искажений и легко выявляются статистическим анализом.

Методы частотной области считаются более устойчивыми. В основе формата *JPEG* лежит дискретное косинусное преобразование (*DCT*), которое применяется к блокам изображения размером  $8 \times 8$ . В результате получается набор коэффициентов, описывающих вклад различных частот. Низкие частоты отражают общую яркость и крупные детали, высокие отвечают за мелкие детали и шум, а средние частоты подходят для встраивания информации, так как изменения в них практически незаметны человеческому глазу.

Алгоритм сокрытия сообщений в *JPEG* строится следующим образом. Изображение преобразуется в цветовую модель *YCbCr*, где наиболее удобным для работы является яркостный канал *Y*. Этот канал разбивается на блоки  $8 \times 8$ , и для каждого блока вычисляется *DCT*. Полученные коэффициенты подвергаются квантованию — делению на элементы матрицы квантования. После этого в выбранный коэффициент блока встраивается один бит сообщения, что реализуется заменой младшего бита величины коэффициента на значение скрываемого бита. Такой приём позволяет спрятать ровно один бит в каждом блоке, что ограничивает вместимость, но обеспечивает устойчивость и скрытность. После внедрения всего сообщения выполняется обратное квантование и обратное преобразование, а изображение собирается обратно и сохраняется в формате *JPEG* с той же таблицей квантования.

Извлечение информации выполняется в обратном порядке. Программа открывает изображение, снова выполняет *DCT* и квантование блоков, считывает младшие биты выбранных коэффициентов и восстанавливает из них последовательность, которая затем интерпретируется как текст в кодировке *UTF-8*. Чтобы извлечение было возможным без дополнительных параметров, в первые 32 бита сообщения записывается его длина в байтах. Таким образом, программа всегда знает, сколько информации нужно считать, и может корректно восстановить сообщение, даже если оно содержит символы *Unicode*.

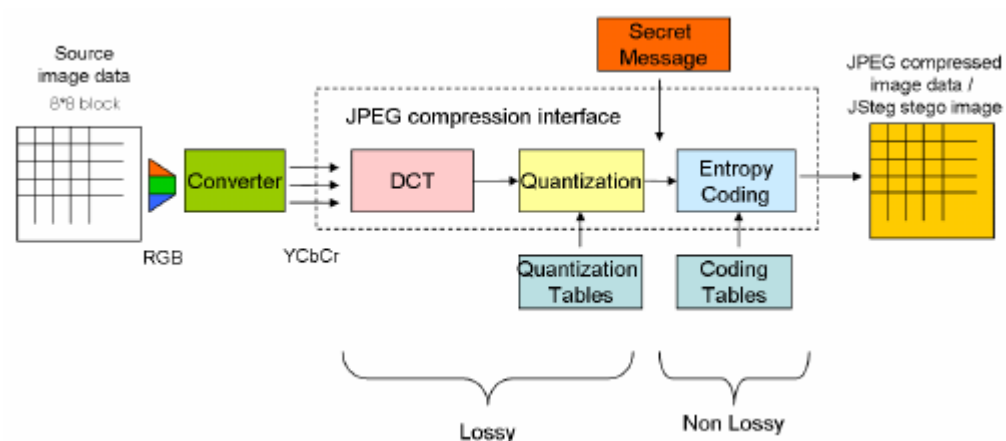


Рисунок 1 – Схема алгоритма сокрытия сообщений в *JPEG*

Данный подход обеспечивает баланс между простотой реализации и устойчивостью к сжатию. Изменения в изображении визуально незаметны, а сообщение корректно восстанавливается при повторном открытии и сохранении файла в *JPEG*.

### 3 ХОД РАБОТЫ

Программа была реализована на языке *Python*. Для работы использовались модули *numpy* и *Pillow*. В папке лабораторной работы размещены четыре основных файла: *input.txt*, содержащий сообщение, которое требуется спрятать; *cover.jpg*, исходное изображение; *stego.jpg*, изображение с внедрённым сообщением; и *output.txt*, куда программа записывает извлечённый текст. Основной алгоритм реализован в файле *main.py*.

Для сокрытия сообщения используется команда *python main.py encrypt*. Программа открывает исходное изображение *cover.jpg*, считывает содержимое файла *input.txt*, преобразует изображение в цветовую модель *YCbCr* и выполняет разбиение яркостного канала на блоки  $8 \times 8$ . В каждый блок встраивается один бит сообщения, для чего модифицируется выбранный коэффициент после квантования. По завершении работы формируется новое изображение *stego.jpg*, которое визуально почти не отличается от исходного. В процессе работы программа выводит в консоль подробный журнал: размер текста, количество использованных блоков, значения коэффициентов и предварительный просмотр встроенных битов. Это позволяет проверить корректность работы алгоритма и убедиться, что сообщение действительно встроено.

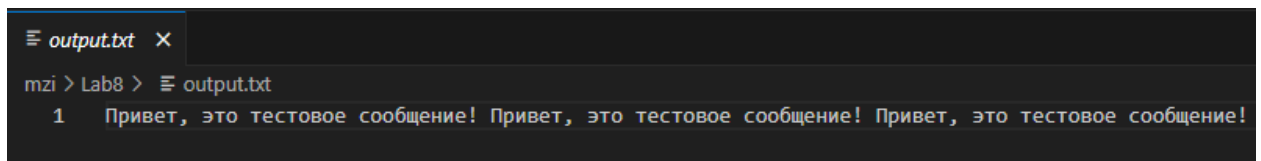
```
[venv] PS D:\bsuir_labs\mzi\lab8> python main.py encrypt
2025-09-25 13:27:04 INFO STEP 1/5 - вход: cover=cover.jpg (3840x2160 px), input=input.txt
2025-09-25 13:27:04 INFO     вместимость: 129600 бит (по 1 биту на блок 8х8)
2025-09-25 13:27:04 INFO STEP 2/5 - подготовка: длина текста=173 байт, будет записано 1416 бит (с длиной).
2025-09-25 13:27:04 INFO     канав-Y, coef=(4,3), QV=28
2025-09-25 13:27:04 INFO     preview(bits): 0000000000000000000000000000000000010101101...101110000110100001011010100100001 | 1011010111010000101111011010000101110001010000101010100100001
2025-09-25 13:27:05 INFO     встраивание: использовано блоков=1416 из 129600 (1.09%), окислено нулей=1416, смена чётности=883
2025-09-25 13:27:05 INFO STEP 3/5 - сохранение: stego.jpg (qtables=DG-75, subsampling=4:4:4)
2025-09-25 13:27:05 INFO STEP 4/5 - самопроверка: длина=173 байт + всего 1416 бит
2025-09-25 13:27:05 INFO     bits(check): 0000000000000000000000000000000000010101101...1011100001010000101010100100001 | 101101011101000010111101101000010110001010000101010100100001
2025-09-25 13:27:05 INFO     text preview: "Привет, это тестовое сообщение! Привет, это тестовое сообщение! Привет, это тестовое сообщение!"
2025-09-25 13:27:05 INFO     результат : OK
2025-09-25 13:27:05 INFO STEP 5/5 - готово.
OK: input.txt -> stego.jpg
```

Рисунок 2 – Скрытие сообщения в изображении

Для извлечения сообщения используется команда *python main.py decrypt*. Программа открывает изображение *stego.jpg*, извлекает из первых 32 бит длину текста, после чего считывает нужное количество коэффициентов и собирает из них сообщение. Полученный результат записывается в файл *output.txt*. В консоль также выводится информация о процессе: считанная длина, количество обработанных блоков, предварительный просмотр битов и текста.

[illegible]

Рисунок 3 – Извлечение сообщения из изображения



```
mzi > Lab8 >   
1  Привет, это тестовое сообщение! Привет, это тестовое сообщение! Привет, это тестовое сообщение!
```

Рисунок 4 – Результат выполнения программы

В качестве теста было использовано сообщение «Привет, это тестовое сообщение! Привет, это тестовое сообщение! Привет, это тестовое сообщение!». Оно было помещено в файл *input.txt*. После выполнения команды *encrypt* программа сохранила изображение *stego.jpg*. Далее выполнение команды *decrypt* позволило извлечь сообщение, и в файле *output.txt* оказалось то же самое содержание. Совпадение входных и выходных данных подтверждает корректность работы программы. Скриншоты работы программы и содержимого файлов приведены в отчёте.

## ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы было подробно рассмотрено понятие стеганографии и её отличие от криптографии. Если криптография делает акцент на защите содержимого информации, то стеганография ориентирована на маскировку самого факта её передачи. На практике это позволяет использовать изображения, аудио- и видеофайлы как скрытые контейнеры для передачи сообщений.

Основное внимание было уделено методам частотной области, которые применяются в формате *JPEG*. Изображение представляется в виде блоков  $8 \times 8$ , для которых вычисляются коэффициенты дискретного косинусного преобразования. Благодаря квантованию можно получить компактное представление изображения и при этом использовать отдельные коэффициенты для встраивания дополнительных данных. Этот метод более устойчив по сравнению с методами пространственной области, так как внесённые изменения малозаметны для человеческого глаза и сохраняются при стандартных преобразованиях изображения.

В практической части работы был реализован алгоритм сокрытия и извлечения сообщений с использованием языка *Python* и библиотек *numpy* и *Pillow*. Программа выполняет чтение текста из файла, преобразует изображение, внедряет сообщение в квантованные коэффициенты и сохраняет результат в новый *JPEG*-файл. Для проверки корректности предусмотрено автоматическое извлечение встроенного текста и сравнение его с исходным. Было подтверждено, что сообщение корректно встраивается и восстанавливается даже при использовании символов *Unicode*.

Программа имеет удобный консольный интерфейс и минимальные требования для запуска. При тестировании сообщение, сохранённое в *input.txt*, было успешно встроено в изображение *cover.jpg* и корректно восстановлено в файле *output.txt*. Внешне изображения до и после встраивания не отличаются, что говорит о скрытности метода.

Таким образом, цель лабораторной работы достигнута. Были изучены теоретические основы стеганографии в частотной области, рассмотрены особенности работы формата *JPEG* и реализован рабочий прототип программы для сокрытия и восстановления текстовых сообщений. Полученный опыт можно использовать для более сложных задач, таких как защита авторских прав (цифровые водяные знаки) или организация скрытых каналов связи.



## ПРИЛОЖЕНИЕ А

### (обязательное)

#### Листинг программного кода

```
import logging
import os
import sys
import math
import warnings
import numpy as np
from PIL import Image

warnings.filterwarnings("ignore", message=".*'mode' parameter is
deprecated.*")

INPUT_TXT = "input.txt"
COVER_JPG = "cover.jpg"
STEGO_JPG = "stego.jpg"
OUTPUT_TXT = "output.txt"

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

BLOCK = 8
CHANNEL_IDX = 0
COEF_POS = (4, 3)
QUALITY = 75

QY = np.array(
    [
        [8, 6, 5, 8, 12, 20, 26, 31],
        [6, 6, 7, 10, 13, 29, 30, 28],
        [7, 7, 8, 12, 20, 29, 35, 28],
        [7, 9, 12, 15, 26, 44, 40, 31],
        [9, 12, 19, 28, 34, 55, 52, 39],
        [12, 18, 28, 32, 41, 52, 57, 46],
        [25, 32, 39, 44, 52, 61, 60, 51],
        [36, 46, 48, 49, 56, 50, 52, 50],
    ],
    dtype=np.int32,
)
QC = np.array(
    [
        [9, 9, 12, 24, 50, 50, 50, 50],
        [9, 11, 13, 33, 50, 50, 50, 50],
        [12, 13, 28, 50, 50, 50, 50, 50],
        [24, 33, 50, 50, 50, 50, 50, 50],
        [50, 50, 50, 50, 50, 50, 50, 50],
        [50, 50, 50, 50, 50, 50, 50, 50],
        [50, 50, 50, 50, 50, 50, 50, 50],
        [50, 50, 50, 50, 50, 50, 50, 50],
    ],
    dtype=np.int32,
)

def _dct_matrix(N: int) -> np.ndarray:
    C = np.zeros((N, N), dtype=np.float64)
    for k in range(N):
        alpha = math.sqrt(1.0 / N) if k == 0 else math.sqrt(2.0 / N)
```

```

        for n in range(N):
            C[k, n] = alpha * math.cos(math.pi * (2 * n + 1) * k / (2 * N))
    return C

_DCT = _dct_matrix(BLOCK)

def dct2(block: np.ndarray) -> np.ndarray:
    return _DCT @ block @ _DCT.T

def idct2(coeff: np.ndarray) -> np.ndarray:
    return _DCT.T @ coeff @ _DCT

def pad_to_multiple(arr: np.ndarray, block: int = BLOCK):
    h, w = arr.shape
    ph = (block - (h % block)) % block
    pw = (block - (w % block)) % block
    if ph == 0 and pw == 0:
        return arr, h, w
    return np.pad(arr, ((0, ph), (0, pw)), mode="edge"), h, w

def unpad(arr: np.ndarray, h: int, w: int) -> np.ndarray:
    return arr[:h, :w]

def text_to_bits(text: str) -> str:
    data = text.encode("utf-8")
    length = len(data)
    len_bits = f"{length:032b}"
    data_bits = "".join(f"{b:08b}" for b in data)
    return len_bits + data_bits

def bits_to_text(bits: str) -> str:
    if len(bits) < 32:
        return ""
    length = int(bits[:32], 2)
    need = 32 + length * 8
    data_bits = bits[32:need]
    bytelist = [int(data_bits[i : i + 8], 2) for i in range(0,
len(data_bits), 8)]
    try:
        return bytes(bytelist).decode("utf-8", errors="strict")
    except UnicodeDecodeError:
        return bytes(bytelist).decode("utf-8", errors="replace")

def capacity_in_bits(h: int, w: int) -> int:
    H = h + (BLOCK - h % BLOCK) % BLOCK
    W = w + (BLOCK - w % BLOCK) % BLOCK
    return (H // BLOCK) * (W // BLOCK)

def lsb_set_for_coeff(c: int, bit: int) -> tuple[int, int, int]:
    was_zero = c == 0
    if c == 0:
        c = 1
    mag = abs(c)
    changed_parity = 0
    if (mag & 1) != bit:
        mag += 1
        changed_parity = 1
    newc = mag if c > 0 else -mag
    return newc, was_zero, changed_parity

def lsb_get_from_coeff(c: int) -> int:
    if c == 0:
        return 0
    return abs(c) & 1

```

```

def to_zigzag_list(Q: np.ndarray) -> list:
    order = [
        (0, 0),
        (0, 1),
        (1, 0),
        (2, 0),
        (1, 1),
        (0, 2),
        (0, 3),
        (1, 2),
        (2, 1),
        (3, 0),
        (4, 0),
        (3, 1),
        (2, 2),
        (1, 3),
        (0, 4),
        (0, 5),
        (1, 4),
        (2, 3),
        (3, 2),
        (4, 1),
        (5, 0),
        (6, 0),
        (5, 1),
        (4, 2),
        (3, 3),
        (2, 4),
        (1, 5),
        (0, 6),
        (0, 7),
        (1, 6),
        (2, 5),
        (3, 4),
        (4, 3),
        (5, 2),
        (6, 1),
        (7, 0),
        (7, 1),
        (6, 2),
        (5, 3),
        (4, 4),
        (3, 5),
        (2, 6),
        (1, 7),
        (2, 7),
        (3, 6),
        (4, 5),
        (5, 4),
        (6, 3),
        (7, 2),
        (7, 3),
        (6, 4),
        (5, 5),
        (4, 6),
        (3, 7),
        (4, 7),
        (5, 6),
        (6, 5),
        (7, 4),
        (7, 5),
        (6, 6),
        (5, 7),
    ]

```

```

        (6, 7),
        (7, 6),
        (7, 7),
    ]
    return [int(Q[i, j]) for (i, j) in order]

def _bits_preview(bits: str, k: int = 32) -> str:
    if not bits:
        return ""
    if len(bits) <= 2 * k:
        return bits
    return bits[:k] + " ... " + bits[-k:]

def _text_preview(s: str, k: int = 120) -> str:
    s1 = s.replace("\n", "\\n")
    return s1 if len(s1) <= k else s1[:k] + "..."

def embed_bits_into_image(img_rgb: Image.Image, bits: str) ->
tuple[Image.Image, dict]:
    ycbcr = img_rgb.convert("YCbCr")
    arr = np.array(ycbcr).astype(np.float64)
    chan = arr[:, :, CHANNEL_IDX]

    chan_pad, H0, W0 = pad_to_multiple(chan, BLOCK)
    H, W = chan_pad.shape
    total_blocks = (H // BLOCK) * (W // BLOCK)

    if len(bits) > total_blocks:
        raise ValueError(
            f"Недостаточно блоков: нужно {len(bits)}, доступно
{total_blocks}."
        )

    outQ = np.zeros_like(chan_pad)
    u, v = COEF_POS
    q = QY

    used_blocks = 0
    zeros_fixed = 0
    parity_changed = 0

    bit_idx = 0
    for by in range(0, H, BLOCK):
        for bx in range(0, W, BLOCK):
            block = chan_pad[by : by + BLOCK, bx : bx + BLOCK] - 128.0
            D = dct2(block)
            Cq = np rint(D / q).astype(np.int32)

            if bit_idx < len(bits):
                b = 1 if bits[bit_idx] == "1" else 0
                target_u, target_v = (u, v) if not (u == 0 and v == 0) else
(0, 1)
                newc, wz, pc = lsb_set_for_coeff(int(Cq[target_u, target_v]),
b)
                Cq[target_u, target_v] = newc
                zeros_fixed += 1 if wz else 0
                parity_changed += 1 if pc else 0
                bit_idx += 1
                used_blocks += 1

            Dm = (Cq * q).astype(np.float64)
            block_rec = idct2(Dm) + 128.0
            outQ[by : by + BLOCK, bx : bx + BLOCK] = block_rec

```

```

outY = np.clip(np rint(outQ), 0, 255).astype(np.uint8)
outY = unpad(outY, H0, W0)

arr[:H0, :W0, CHANNEL_IDX] = outY
out_ycbcr = Image.fromarray(arr[:H0, :W0].astype(np.uint8), mode="YCbCr")
out_rgb = out_ycbcr.convert("RGB")

stats = {
    "total_blocks": total_blocks,
    "used_blocks": used_blocks,
    "zeros_fixed": zeros_fixed,
    "parity_changed": parity_changed,
    "qy_uv": int(q[u, v]),
    "coef_pos": (u, v),
}
return out_rgb, stats

def extract_bits_from_image(img_rgb: Image.Image, need_bits: int) -> str:
    ycbcr = img_rgb.convert("YCbCr")
    arr = np.array(ycbcr).astype(np.float64)
    chan = arr[:, :, CHANNEL_IDX]

    chan_pad, H0, W0 = pad_to_multiple(chan, BLOCK)
    H, W = chan_pad.shape
    total_blocks = (H // BLOCK) * (W // BLOCK)
    if need_bits > total_blocks:
        raise ValueError(f"Запрошено {need_bits} бит, доступно
{total_blocks}.")

    bits = []
    u, v = COEF_POS
    q = QY
    bit_idx = 0
    for by in range(0, H, BLOCK):
        for bx in range(0, W, BLOCK):
            block = chan_pad[by : by + BLOCK, bx : bx + BLOCK] - 128.0
            D = dct2(block)
            Cq = np rint(D / q).astype(np.int32)
            c = Cq[u, v] if not (u == 0 and v == 0) else Cq[0, 1]
            bits.append("1" if lsb_get_from_coeff(int(c)) == 1 else "0")
            bit_idx += 1
            if bit_idx >= need_bits:
                break
        if bit_idx >= need_bits:
            break
    return "".join(bits)

def save_jpeg_with_qtables(img_rgb: Image.Image, path: str):
    qtables = [to_zigzag_list(QY), to_zigzag_list(QC)]
    img_rgb.save(
        path,
        format="JPEG",
        qtables=qtables,
        subsampling=0,
        quality=QUALITY,
        optimize=False,
    )

def cmd_encrypt():
    if not os.path.exists(COVER_JPG):
        raise SystemExit(
            f"Нет {COVER_JPG}. Положи исходное изображение (JPEG) рядом со
скриптом."
        )

```

```

with open(INPUT_TXT, "r", encoding="utf-8") as f:
    text = f.read()
    img = Image.open(COVER_JPG).convert("RGB")
    w, h = img.size
    cap = capacity_in_bits(h, w)
    log.info(
        "STEP 1/5 - вход: cover=%s (%dx%d px), input=%s", COVER_JPG, w, h,
INPUT_TXT
    )
    log.info("                вместимость: %d бит (по 1 биту на блок 8x8)", cap)

    bits = text_to_bits(text)
    log.info(
        "STEP 2/5 - подготовка: длина текста=%d байт, будет записано %d бит
(с длиной).",
        len(text.encode("utf-8")),
        len(bits),
    )
    if len(bits) > cap:
        raise SystemExit(
            f"Сообщение слишком длинное: нужно {len(bits)} бит, доступно
{cap}."
        )
    log.info(
        "                канал=Y, coef=(%d,%d), QY=%d",
        COEF_POS[0],
        COEF_POS[1],
        int(QY[COEF_POS]),
    )
    log.info(
        "                preview(bits): %s | %s",
        _bits_preview(bits),
        _bits_preview(bits[-64:]),
    )

    stego_rgb, stats = embed_bits_into_image(img, bits)
    log.info(
        "                встраивание: использовано блоков=%d из %d (%.2f%),
оживлено нулей=%d, смена чётности=%d",
        stats["used_blocks"],
        stats["total_blocks"],
        100.0 * stats["used_blocks"] / stats["total_blocks"],
        stats["zeros_fixed"],
        stats["parity_changed"],
    )

    save_jpeg_with_qtables(stego_rgb, STEGO_JPG)
    log.info(
        "STEP 3/5 - сохранение: %s (qtables=IJG~%d, subsampling=4:4:4)",
        STEGO_JPG,
        QUALITY,
    )

    stego_check = Image.open(STEGO_JPG).convert("RGB")
    length_bits = extract_bits_from_image(stego_check, 32)
    msg_len = int(length_bits, 2)
    total_needed = 32 + msg_len * 8
    log.info(
        "STEP 4/5 - самопроверка: длина=%d байт → всего %d бит", msg_len,
total_needed
    )
    bits_back = extract_bits_from_image(stego_check, total_needed)
    txt_back = bits_to_text(bits_back)
    ok = txt_back == text

```

```

log.info(
    "          bits(check) : %s | %s",
    _bits_preview(bits_back),
    _bits_preview(bits_back[-64:]),
)
log.info('          text preview: "%s"', _text_preview(txt_back))
log.info("          результат      : %s", "OK" if ok else "MISMATCH")

log.info("STEP 5/5 - готово.")
print(f"OK: {INPUT_TXT} -> {STEGO_JPG}")

def cmd_decrypt():
    if not os.path.exists(STEGO_JPG):
        raise SystemExit(f"Нет {STEGO_JPG}. Сначала запусти encrypt.")
    img = Image.open(STEGO_JPG).convert("RGB")
    w, h = img.size
    log.info("STEP 1/3 - вход: stego=%s (%dx%d px)", STEGO_JPG, w, h)

    len_bits = extract_bits_from_image(img, 32)
    msg_len = int(len_bits, 2)
    total_bits = 32 + msg_len * 8
    log.info(
        "STEP 2/3 - извлечение: длина=%d байт (len_bits=%s), всего %d бит",
        msg_len,
        _bits_preview(len_bits),
        total_bits,
    )

    bits = extract_bits_from_image(img, total_bits)
    text = bits_to_text(bits)
    log.info(
        "          bits preview : %s | %s",
        _bits_preview(bits),
        _bits_preview(bits[-64:]),
    )
    log.info('          text preview : "%s"', _text_preview(text))

    with open(OUTPUT_TXT, "w", encoding="utf-8") as f:
        f.write(text)
    log.info("STEP 3/3 - записано в %s", OUTPUT_TXT)
    print(f"OK: {STEGO_JPG} -> {OUTPUT_TXT}")

def main():
    if len(sys.argv) != 2 or sys.argv[1] not in ("encrypt", "decrypt"):
        print("Использование:")
        print("  python main.py encrypt    прочитать input.txt + cover.jpg -> stego.jpg")
        print("  python main.py decrypt    прочитать stego.jpg -> output.txt")
        raise SystemExit(1)
    cmd = sys.argv[1]
    if cmd == "encrypt":
        cmd_encrypt()
    elif cmd == "decrypt":
        cmd_decrypt()

if __name__ == "__main__":
    main()

```