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](#_Toc209095978)

[2 Теоретеческие сведения 4](#_Toc209095979)

[3 Ход работы 6](#_Toc209095980)

[Заключение 8](#_Toc209095981)

[Приложение А (обязательное) Листинг программного кода 9](#_Toc209095982)

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 8×8, 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 8×8. 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 8×8, 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*.

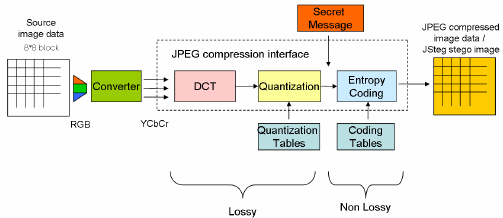


Рисунок 1 – Схема алгоритма сокрытия сообщений в *JPEG*

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 8×8. 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.

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2 – Сокрытие 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.

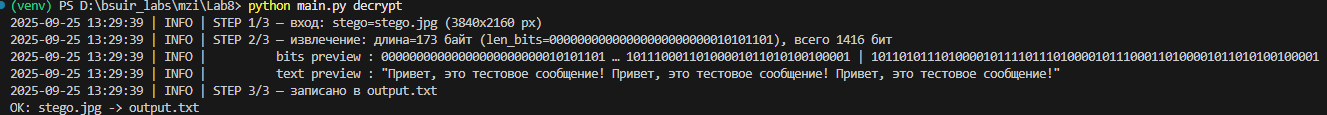


Рисунок 3 – Извлечение сообщения из изображения

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

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 8×8, 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 скрытия 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 8×8)", 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()