Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра ПМиК

Лабораторная работа 2

по дисциплине «Прикладная стеганография»

Выполнил: ст. гр. ЗМП-41 Лёвкин И. А.

Проверила: Мерзлякова Е.Ю.

Новосибирск 2025

**Цифровой водяной знак**

Программа реализует метод внедрения цифрового водяного знака (ЦВЗ) в изображения с использованием модификации синего канала. Основные принципы метода:

1. **Преобразование данных**: водяной знак (текст или файл) преобразуется в последовательность битов.
2. **Выбор пикселей**: для внедрения используются случайно выбранные пиксели внутри изображения (с исключением граничных областей).
3. **Встраивание**: Каждый бит водяного знака встраивается в синий канал выбранного пикселя с учетом его яркости:
   * Модификация синего канала пропорционально яркости пикселя
4. **Извлечение**: для извлечения используется предсказание исходного значения синего канала на основе соседних пикселей и анализ разницы между фактическим и предсказанным значением.

**Основные функции программы**

Программа имеет два основных режима работы, реализованных в виде вкладок:

Вкладка "Внедрение ЦВЗ"

* Загрузка исходного изображения (формат BMP)
* Ввод текста водяного знака или загрузка из файла
* Генерация и отображение ключей (координат пикселей с водяными знаками)
* Сохранение модифицированного изображения

Вкладка "Извлечение ЦВЗ"

* Загрузка изображения с водяным знаком
* Ввод ключей (координат пикселей)
* Извлечение и отображение скрытого текста

3. Особенности реализации

1. **Устойчивость**: использование яркости пикселя при модификации делает водяной знак менее заметным в темных областях изображения.
2. **Прогнозирование**: при извлечении используется предсказание исходного значения синего канала на основе соседних пикселей, что повышает точность извлечения.

**Оценка алгоритма**

Проведём встраивание данных в изображение на 8-битном изображении с палитрой из оттенков серого. Встраиваемый текст на английском языке и составляет размер 16 КБ. Размер контейнера в свою очередь составляет 257 КБ.

Мы встроили текст размером в 16 КБ. При визуальной оценке контейнера заметно что в него встроено сообщение.

|  |  |
| --- | --- |
| Рис. 1. Контейнер без сообщения | Рис. 2. Контейнер после встраивания сообщения |

**Листинг**

import sys

import os

from PyQt6.QtWidgets import (

    QApplication,

    QMainWindow,

    QTabWidget,

)

from embed\_tab import EmbedTab

from extract\_tab import ExtractTab

def main():

    app = QApplication(sys.argv)

    window = WatermarkApp()

    window.show()

    sys.exit(app.exec())

class WatermarkApp(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Цифровой водяной знак")

        self.setGeometry(100, 100, 1280, 720)

        self.setMinimumSize(800, 600)

        self.\_setup\_ui()

    def \_setup\_ui(self):

        self.tabs = QTabWidget()

        self.setCentralWidget(self.tabs)

        self.embed\_tab = EmbedTab()

        self.extract\_tab = ExtractTab()

        self.tabs.addTab(self.embed\_tab, "Внедрение ЦВЗ")

        self.tabs.addTab(self.extract\_tab, "Извлечение ЦВЗ")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

from PyQt6.QtWidgets import QMessageBox, QLabel, QWidget

from PyQt6.QtGui import QPixmap, QImage

from PIL import Image

def load\_image(path) -> QPixmap:

    img = Image.open(path)

    img.thumbnail((512, 512))

    if img.mode == "RGB":

        rgb\_image = img.convert("RGB")

        qimage = QImage(

            rgb\_image.tobytes(),

            rgb\_image.size[0],

            rgb\_image.size[1],

            QImage.Format.Format\_RGB888,

        )

    else:

        qimage = QImage(path)

    return QPixmap.fromImage(qimage)

from PyQt6.QtWidgets import QApplication, QTextEdit

from PyQt6.QtCore import QTimer

class SafeTextEdit(QTextEdit):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.\_text\_queue = ""

        self.\_chunk\_size = 10000

        self.\_timer = QTimer(self)

        self.\_timer.timeout.connect(self.\_process\_chunk)

    def set\_large\_text(self, text):

        self.\_text\_queue = text

        self.clear()

        self.\_timer.start(5)

    def \_process\_chunk(self):

        if not self.\_text\_queue:

            self.\_timer.stop()

            return

        chunk = self.\_text\_queue[: self.\_chunk\_size]

        self.\_text\_queue = self.\_text\_queue[self.\_chunk\_size :]

        self.append(chunk)  # Добавляем по частям

        QApplication.processEvents()  # Обрабатываем другие события

from PyQt6.QtWidgets import (

    QWidget,

    QVBoxLayout,

    QHBoxLayout,

    QLabel,

    QLineEdit,

    QPushButton,

    QTextEdit,

    QFileDialog,

    QMessageBox,

    QScrollArea,

)

from PyQt6.QtGui import QPixmap, QImage

from PyQt6.QtCore import Qt

from PIL import Image

import digital\_watermark

import utils

from image\_label import ImageLabel

from safe\_text\_edit import SafeTextEdit

class ExtractTab(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.\_setup\_ui()

    def \_setup\_ui(self):

        main\_layout = QHBoxLayout(self)

        # Левая панель (управление)

        left\_panel = QWidget()

        left\_layout = QVBoxLayout(left\_panel)

        # Панель управления

        control\_frame = QWidget()

        control\_layout = QVBoxLayout(control\_frame)

        # Выбор изображения

        image\_layout = QHBoxLayout()

        image\_layout.addWidget(QLabel("Файл изображения:"))

        self.image\_path\_entry = QLineEdit()

        image\_layout.addWidget(self.image\_path\_entry)

        browse\_btn = QPushButton("Обзор")

        browse\_btn.clicked.connect(self.browse\_image)

        image\_layout.addWidget(browse\_btn)

        control\_layout.addLayout(image\_layout)

        # Ключи

        keys\_layout = QHBoxLayout()

        keys\_layout.addWidget(QLabel("Ключи:"))

        self.keys\_entry = SafeTextEdit()

        keys\_layout.addWidget(self.keys\_entry)

        control\_layout.addLayout(keys\_layout)

        # Кнопка извлечения

        self.extract\_button = QPushButton("Извлечь ЦВЗ")

        self.extract\_button.clicked.connect(self.extract)

        control\_layout.addWidget(self.extract\_button)

        left\_layout.addWidget(control\_frame)

        # Результат (извлеченный текст)

        result\_frame = QWidget()

        result\_layout = QVBoxLayout(result\_frame)

        result\_layout.addWidget(QLabel("Извлеченный текст:"))

        self.extracted\_text = SafeTextEdit()

        self.extracted\_text.setReadOnly(True)

        result\_layout.addWidget(self.extracted\_text)

        left\_layout.addWidget(result\_frame)

        left\_layout.addStretch()

        main\_layout.addWidget(left\_panel)

        # Правая панель (изображение)

        right\_panel = QWidget()

        right\_panel.setMinimumSize(512, 512)

        right\_layout = QVBoxLayout(right\_panel)

        self.image\_label = ImageLabel()

        scroll = QScrollArea()

        scroll.setWidget(self.image\_label)

        scroll.setWidgetResizable(True)

        right\_layout.addWidget(scroll)

        main\_layout.addWidget(right\_panel)

    def extract(self):

        image\_path = self.image\_path\_entry.text()

        if not image\_path:

            QMessageBox.critical(self, "Ошибка", "Выберите изображение")

            return

        try:

            keys\_string = self.keys\_entry.toPlainText().replace("\n", "")

            if not keys\_string:

                QMessageBox.critical(self, "Ошибка", "Введите ключи")

                return

            numbers = list(map(int, keys\_string.split(",")))

            if len(numbers) % 2 != 0:

                QMessageBox.critical(

                    self,

                    "Ошибка",

                    "Неверный формат ключей. Ожидались пары чисел",

                )

                return

            coord\_pairs = list(zip(numbers[::2], numbers[1::2]))

            extracted\_bits = digital\_watermark.extract\_watermark(

                image\_path, coord\_pairs

            )

            bit\_string = "".join(map(str, extracted\_bits))

            bytes\_list = [

                int(bit\_string[i : i + 8], 2) for i in range(0, len(bit\_string), 8)

            ]

            extracted\_text = bytes(bytes\_list).decode("utf-8", errors="replace")

            self.extracted\_text.set\_large\_text(extracted\_text or "Текст не найден")

        except Exception as e:

            QMessageBox.critical(self, "Ошибка", f"Ошибка при извлечении:\n{str(e)}")

    def browse\_image(self):

        filepath, \_ = QFileDialog.getOpenFileName(

            self, "Выберите изображение", "", "Images (\*.bmp);;All Files (\*)"

        )

        if filepath:

            self.image\_path\_entry.setText(filepath)

            self.display\_image(filepath)

    def display\_image(self, path):

        try:

            self.image\_label.setPixmap(utils.load\_image(path))

        except Exception as e:

            QMessageBox.critical(

                self, "Ошибка", f"Не удалось загрузить изображение:\n{str(e)}"

            )

import os

from PyQt6.QtWidgets import (

    QApplication,

    QWidget,

    QVBoxLayout,

    QHBoxLayout,

    QLabel,

    QLineEdit,

    QPushButton,

    QFileDialog,

    QMessageBox,

    QScrollArea,

)

import digital\_watermark

import utils

from image\_label import ImageLabel

from safe\_text\_edit import SafeTextEdit

class EmbedTab(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.image\_path = ""

        self.watermark\_path = ""

        self.\_setup\_ui()

    def \_setup\_ui(self):

        main\_layout = QHBoxLayout(self)

        # Левая панель (управление)

        left\_panel = QWidget()

        left\_layout = QVBoxLayout(left\_panel)

        # Панель управления

        control\_frame = QWidget()

        control\_layout = QVBoxLayout(control\_frame)

        # Выбор изображения

        image\_layout = QHBoxLayout()

        image\_layout.addWidget(QLabel("Изображение:"))

        self.image\_path\_entry = QLineEdit()

        image\_layout.addWidget(self.image\_path\_entry)

        browse\_image\_btn = QPushButton("Обзор")

        browse\_image\_btn.clicked.connect(self.browse\_image)

        image\_layout.addWidget(browse\_image\_btn)

        control\_layout.addLayout(image\_layout)

        # Ввод ЦВЗ

        watermark\_layout = QHBoxLayout()

        watermark\_layout.addWidget(QLabel("ЦВЗ:"))

        self.watermark\_entry = QLineEdit()

        watermark\_layout.addWidget(self.watermark\_entry)

        browse\_watermark\_btn = QPushButton("Обзор")

        browse\_watermark\_btn.clicked.connect(self.browse\_watermark)

        watermark\_layout.addWidget(browse\_watermark\_btn)

        control\_layout.addLayout(watermark\_layout)

        # Кнопка внедрения

        self.embed\_button = QPushButton("Внедрить ЦВЗ")

        self.embed\_button.clicked.connect(self.embed)

        control\_layout.addWidget(self.embed\_button)

        left\_layout.addWidget(control\_frame)

        keys\_frame = QWidget()

        keys\_layout = QVBoxLayout(keys\_frame)

        keys\_layout.setContentsMargins(0, 0, 0, 0)  # Убираем лишние отступы

        keys\_layout.setSpacing(5)  # Расстояние между элементами

        # Создаем горизонтальный контейнер для label и кнопки

        header\_layout = QHBoxLayout()

        header\_layout.setContentsMargins(0, 0, 0, 0)

        # Добавляем label и кнопку с выравниванием по краям

        label = QLabel("Ключи для извлечения:")

        header\_layout.addWidget(label)

        copy\_btn = QPushButton("Копировать")

        copy\_btn.clicked.connect(self.copy\_to\_clipboard)

        header\_layout.addWidget(copy\_btn)

        # Добавляем растягивающий элемент между label и кнопкой

        header\_layout.addStretch()

        # Добавляем горизонтальный layout в вертикальный

        keys\_layout.addLayout(header\_layout)

        # Создаем текстовое поле, которое займет все оставшееся пространство

        self.keys\_entry = SafeTextEdit()

        self.keys\_entry.setReadOnly(True)

        keys\_layout.addWidget(

            self.keys\_entry, stretch=1

        )  # stretch=1 для заполнения пространства

        left\_layout.addWidget(keys\_frame)

        left\_layout.addStretch()

        main\_layout.addWidget(left\_panel)

        # Правая панель (изображение)

        right\_panel = QWidget()

        right\_panel.setMinimumSize(512, 512)

        right\_layout = QVBoxLayout(right\_panel)

        self.image\_label = ImageLabel()

        scroll = QScrollArea()

        scroll.setWidget(self.image\_label)

        scroll.setWidgetResizable(True)

        right\_layout.addWidget(scroll)

        main\_layout.addWidget(right\_panel)

    def embed(self):

        if not self.image\_path:

            QMessageBox.critical(self, "Ошибка", "Выберите изображение")

            return

        watermark\_bytes = self.load\_watermark\_bytes()

        if not watermark\_bytes:

            QMessageBox.critical(self, "Ошибка", "Введите текст водяного знака")

            return

        save\_path, \_ = QFileDialog.getSaveFileName(

            self,

            "Сохранить изображение",

            "",

            "BMP Files (\*.bmp);",

        )

        if not save\_path:

            return

        try:

            embedded\_img, keys = digital\_watermark.embed\_watermark(

                self.image\_path, watermark\_bytes

            )

            embedded\_img.save(save\_path)

            keys\_str = ",".join([f"{x},{y}" for x, y in keys])

            self.keys\_entry.set\_large\_text(keys\_str)

            QMessageBox.information(self, "Успех", "Водяной знак успешно внедрен")

        except Exception as e:

            QMessageBox.critical(self, "Ошибка", f"Ошибка при внедрении:\n{str(e)}")

    def load\_watermark\_bytes(self):

        if os.path.exists(self.watermark\_path):

            with open(self.watermark\_path, "rb") as f:

                return f.read()

        else:

            return self.watermark\_entry.text().encode("utf-8")

    def browse\_image(self):

        filepath, \_ = QFileDialog.getOpenFileName(

            self, "Выберите изображение", "", "Images (\*.bmp);;All Files (\*)"

        )

        if filepath:

            self.image\_path = filepath

            self.image\_path\_entry.setText(filepath)

            self.display\_image(filepath)

    def browse\_watermark(self):

        filepath, \_ = QFileDialog.getOpenFileName(

            self, "Выберите файл с ЦВЗ", "", "All Files (\*)"

        )

        if filepath:

            self.watermark\_path = filepath

            self.watermark\_entry.setText(filepath)

    def display\_image(self, path):

        try:

            self.image\_label.setPixmap(utils.load\_image(path))

        except Exception as e:

            QMessageBox.critical(

                self, "Ошибка", f"Не удалось загрузить изображение:\n{str(e)}"

            )

    def copy\_to\_clipboard(self):

        text = self.keys\_entry.toPlainText().replace("\n", "")

        if text:

            QApplication.clipboard().setText(text)

            QMessageBox.information(self, "Скопировано", "Ключи скопированы в буфер")

        else:

            QMessageBox.warning(self, "Пусто", "Нет ключей для копирования")

from PIL import Image

import numpy as np

import random

from typing import Tuple, List

PADDING\_PIXELS = 4  # Padding for extraction

def embed\_watermark(

    image\_path: str, watermark\_bytes: bytes, q: float = 0.5, seed: int = 42

) -> Tuple[Image.Image, List[Tuple[int, int]]]:

    img = Image.open(image\_path).convert("RGB")

    img\_array = np.array(img, dtype=np.float32)

    height, width, \_ = img\_array.shape

    # Convert text to binary bits

    watermark\_bits = \_bytes\_to\_bits(watermark\_bytes)

    watermark\_length = len(watermark\_bits)

    # Generate random coordinates for embedding

    pixel\_coords = \_generate\_embedding\_coordinates(

        height, width, watermark\_length, seed

    )

    # Embed each bit into the image

    \_embed\_bits(img\_array, pixel\_coords, watermark\_bits, q)

    # Convert back to image

    img\_array = np.clip(img\_array, 0, 255).astype(np.uint8)

    return Image.fromarray(img\_array), pixel\_coords

def \_bytes\_to\_bits(bytes: str):

    binary\_string = "".join([format(byte, "08b") for byte in bytes])

    return [int(bit) for bit in binary\_string]

def \_generate\_embedding\_coordinates(

    height: int, width: int, required\_count: int, seed: int

) -> List[Tuple[int, int]]:

    random.seed(seed)

    total\_pixels = (height - 2 \* PADDING\_PIXELS) \* (width - 2 \* PADDING\_PIXELS)

    if required\_count > total\_pixels:

        raise ValueError("Watermark is too long for the image size")

    available\_pixels = [

        (y, x)

        for y in range(PADDING\_PIXELS, height - PADDING\_PIXELS)

        for x in range(PADDING\_PIXELS, width - PADDING\_PIXELS)

    ]

    return random.sample(available\_pixels, required\_count)

def \_embed\_bits(

    img\_array: np.ndarray, coords: List[Tuple[int, int]], bits: List[int], q: float

) -> None:

    for i, (y, x) in enumerate(coords):

        R, G, B = img\_array[y, x]

        L = 0.299 \* R + 0.587 \* G + 0.114 \* B  # Luminance

        message\_bit = bits[i]

        # Modify blue channel

        img\_array[y, x, 2] = B + (2 \* message\_bit - 1) \* L \* q

        # Handle overflow

        overflow\_flag = 0

        if img\_array[y, x, 2] > 255:

            img\_array[y, x, 2] = 255

            overflow\_flag = 1

        elif img\_array[y, x, 2] < 0:

            img\_array[y, x, 2] = 0

            overflow\_flag = 1

        # Store overflow flag in LSB of green channel

        img\_array[y, x, 1] = (int(G) & 0xFE) | overflow\_flag

def extract\_watermark(

    image\_path: str, coords: List[Tuple[int, int]], c: int = 2

) -> List[int]:

    img = Image.open(image\_path).convert("RGB")

    img\_array = np.array(img, dtype=np.float32)

    watermark\_bits = []

    for y, x in coords:

        B = img\_array[y, x, 2]

        G = img\_array[y, x, 1]

        overflow\_flag = int(G) & 1  # Get overflow flag from LSB of green channel

        # Predict original B value

        B\_pred = \_predict\_blue\_channel(img\_array, y, x, c)

        # Recover the embedded bit

        if overflow\_flag == 1:

            watermark\_bits.append(1 if B == 255 else 0)

        else:

            watermark\_bits.append(1 if (B - B\_pred) > 0 else 0)

    return watermark\_bits

def \_predict\_blue\_channel(img\_array: np.ndarray, y: int, x: int, c: int) -> float:

    height, width, \_ = img\_array.shape

    neighbors = []

    # Collect vertical neighbors

    for k in range(-c, c + 1):

        if 0 <= y + k < height:

            neighbors.append(img\_array[y + k, x, 2])

    # Collect horizontal neighbors

    for k in range(-c, c + 1):

        if 0 <= x + k < width:

            neighbors.append(img\_array[y, x + k, 2])

    # Remove center pixel (added twice)

    if len(neighbors) >= 2:

        neighbors.remove(img\_array[y, x, 2])

        neighbors.remove(img\_array[y, x, 2])

    return (sum(neighbors) / (4 \* c)) if neighbors else img\_array[y, x, 2]