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]
```