Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Институт информатики и вычислительной техники

Кафедра прикладной математики и кибернетики

# Лабораторная работа №7 по дисциплине

**Прикладная стеганография**

Выполнил:

студент гр.МГ-411 Шевельков П.С.

ФИО студента

«15» мая 2025 г.

Новосибирск 2025 г.

# Задание на лабораторную работу:

Модифицировать метод встраивания, реализованный в рамках работы № 4.

* Программа для встраивания информации должна брать текст из файла, преобразовывать его в двоичный вид, складывать с псевдослучайным ключом и записывать в изображение-контейнер в максимально возможном объеме. Также программа должна извлекать обратно вложенный текст **без ошибок**.
* Модифицировать метод встраивания, применив стеганографический **код, базирующийся на линейной хэш-функции**.
* Вывести для сравнения, сколько бит максимально получилось встроить в один и тот же контейнер до и после модификации.
* Применить **код добавления избыточности,** оценивая распределение вероятностей бит в областях 8х8 пикселей пустого контейнера перед интерполяцией. Метод оценки вероятностей выбрать самостоятельно.
* Провести сравнительный стегоанализ полученных вариантов метода встраивания с помощью реализованного в работе 5 приложения.
* Подготовить **доклад-выступление** о полученных результатах.
* **Отчет по работе** должен содержать описание применяемых стеганографических методов, описание программы и блок-схему ее работы. Также в отчет необходимо включить полученные результаты на 10 различных контейнерах и ссылку на исходный код программы.

# Результаты работы программы:

# 

рисунок 1. Встраивание с помощью хэша.

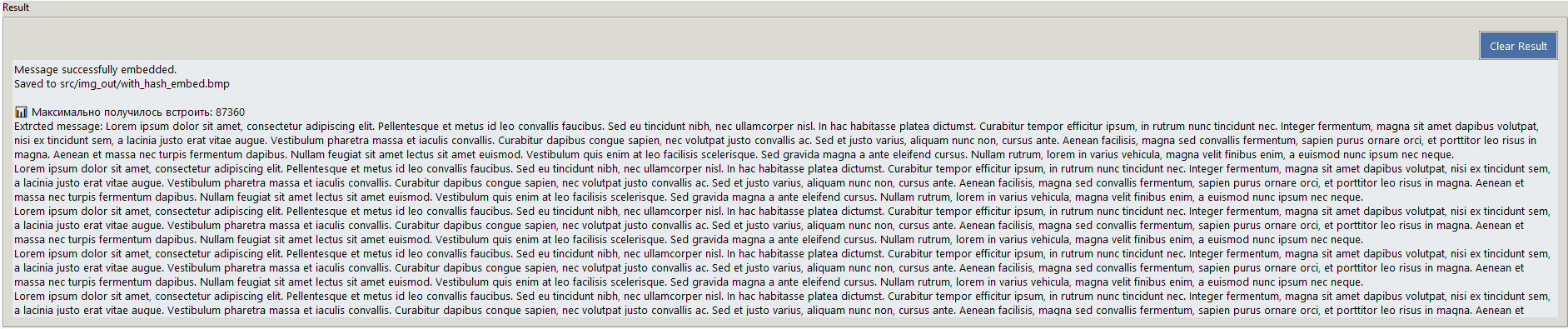


рисунок 2. Извлечение сообщения.

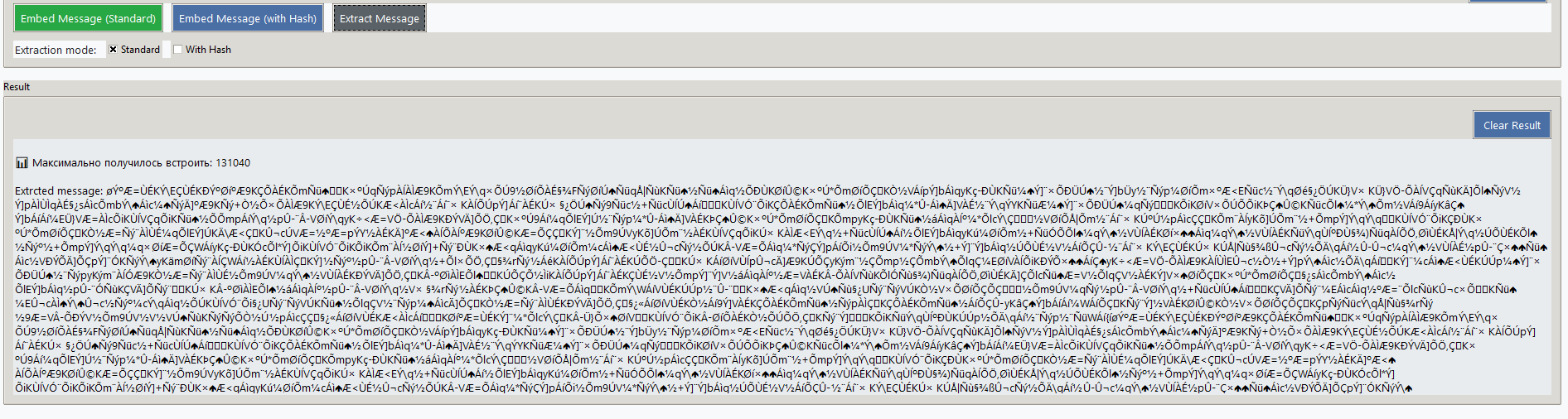


рисунок 3. Попытка извлечь сообщение, зашифрованное с хэшем, обычной интерполяцией.

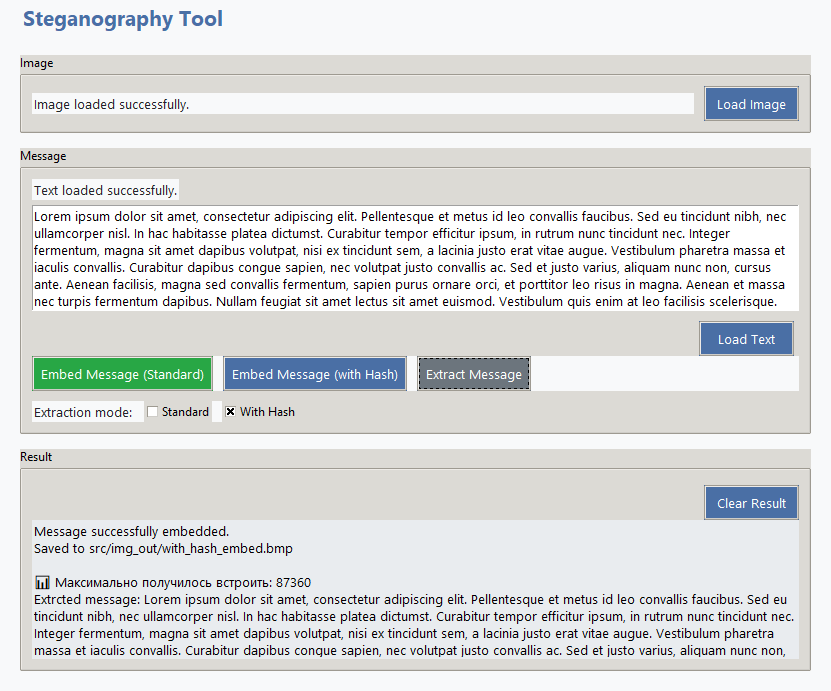


рисунок 4. Первый контейнер 512х512

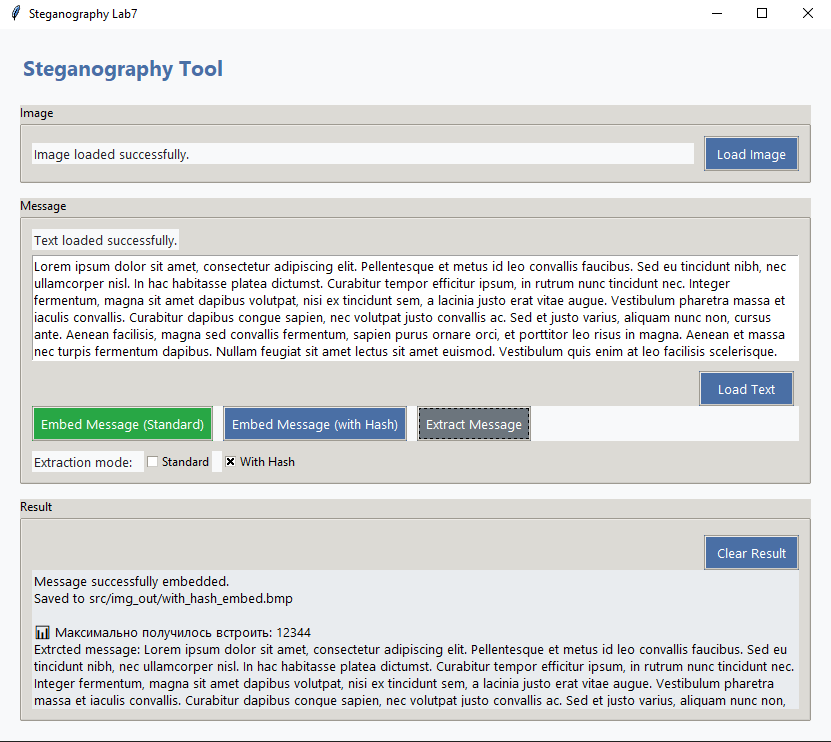


рисунок 4. Второй контейнер 212х175

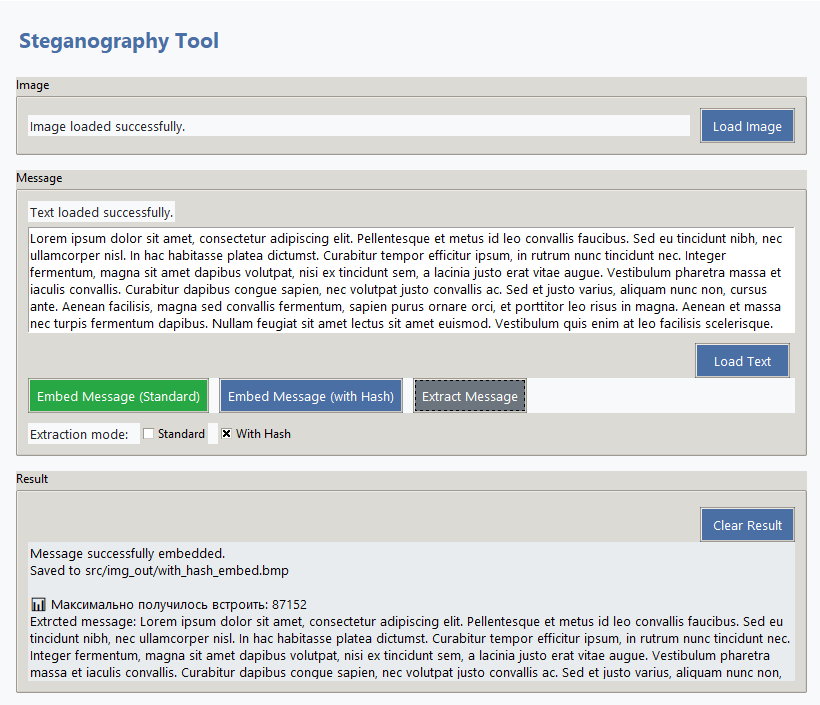


рисунок 5. Третий контейнер 604х433



рисунок 6. Четвертый контейнер 489х750

**Описание программы**

1. **Загрузка изображений**:
   * Программа позволяет загружать изображения в формате BMP для дальнейшей обработки.
2. **Внедрение сообщений**:
   * Реализованы два метода внедрения сообщений:
     + **Стандартный метод**: Внедрение битов сообщения в изображение без дополнительных изменений.
     + **Метод с хешированием**: Внедрение битов сообщения с использованием хеш-функции для повышения устойчивости к обнаружению.
3. **Извлечение сообщений**:
   * Программа поддерживает два метода извлечения:
     + **Стандартный метод**: Извлечение битов сообщения из изображения без дополнительных проверок.
     + **Метод с хешированием**: Извлечение битов сообщения с проверкой целостности с помощью хеш-функции.
4. **Анализ внедрения**:
   * Программа проводит анализ внедрения сообщений, включая проверку целостности и корректности извлечённых данных.
5. **Отображение результатов**:
   * Результаты внедрения и извлечения отображаются в виде текстового отчёта, а также графически для наглядности.
6. **Сохранение результатов**:
   * Возможность сохранить результаты анализа и отчёты в файл для дальнейшего использования.

**Методы внедрения сообщений:**

1. **Стандартный метод внедрения**:
   * Простой и эффективный способ внедрения битов сообщения в изображение, без дополнительных изменений.
2. **Метод с хешированием**:
   * Внедрение битов сообщения с использованием хеш-функции для повышения устойчивости к обнаружению и проверки целостности.

**Архитектура программы:**

1. **MainWindow (главное окно)**:
   * Интерфейс на основе Tkinter.
   * Содержит:
     + Виджет для отображения загруженных изображений.
     + Панель управления (выбор метода внедрения, параметры анализа).
     + Отображение результатов внедрения и извлечения.
   * Поддерживает асинхронную обработку через использование потоков.
2. **SteganographyApp**:
   * Основной класс приложения, инициализирующий интерфейс и обрабатывающий события.
3. **Методы обработки**:
   * extract\_with\_hash: Извлечение битов сообщения с проверкой хеш-функции.
   * embed\_with\_hash: Внедрение битов сообщения с использованием хеширования.
   * extract\_standard: Стандартное извлечение битов сообщения.
   * embed\_message\_standard: Стандартное внедрение битов сообщения.
   * generate\_random\_binary\_matrix: Генерация случайной двоичной матрицы для внедрения.
   * save\_image: Сохранение обработанного изображения.
   * load\_image: Загрузка изображения для обработки.
   * load\_text: Загрузка текстового сообщения для внедрения.
4. **Дополнительные функции**:
   * Методы для преобразования битов в строку и обратно, а также для выполнения операций с битами (например, XOR).

**Выводы:**

1. **Эффективность внедрения**:

Программа демонстрирует высокую эффективность внедрения сообщений в изображения, обеспечивая возможность использования как стандартного метода, так и метода с хешированием. Это позволяет пользователям выбирать подходящий метод в зависимости от требований к безопасности и устойчивости к обнаружению.

1. **Устойчивость к обнаружению**:

Метод с хешированием значительно повышает устойчивость к обнаружению внедрённых сообщений, что делает его предпочтительным для приложений, где важна защита информации.

1. **Анализ и проверка целостности**:

Внедрение механизма проверки целостности с помощью хеш-функции позволяет пользователям уверенно извлекать сообщения, зная, что они не были изменены. Это особенно важно в контексте передачи конфиденциальной информации.

1. **Удобство использования**:

Интуитивно понятный интерфейс на основе Tkinter делает программу доступной для пользователей с различным уровнем подготовки. Возможность загрузки изображений и текстов, а также простота в использовании методов внедрения и извлечения сообщений способствуют широкому применению программы.

1. **Гибкость и расширяемость**:

Архитектура программы позволяет легко добавлять новые методы внедрения и анализа, что открывает возможности для дальнейшего развития и улучшения функциональности. Пользователи могут адаптировать программу под свои нужды, добавляя новые алгоритмы или улучшая существующие.

1. **Практическое применение**:

Программа может быть использована в различных областях, включая защиту авторских прав, безопасную передачу данных и скрытую коммуникацию. Это делает её полезным инструментом как для исследователей, так и для практиков в области информационной безопасности.

## **Ссылка на программу:**

<https://github.com/bothyD/steganograf>

**Листинг:**

import tkinter as tk

from tkinter import filedialog, messagebox

from tkinter import ttk

import numpy as np

from matplotlib import pyplot as plt

from PIL import Image

import random

from bitarray import bitarray

import os

from collections import Counter

class SteganographyApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Steganography Lab7")

        self.container\_image = None

        self.stego\_image = None

        self.text\_input = None

        self.message\_bits = []

        self.key = [1, 0, 1, 1, 0, 1, 0, 0]

        self.A =  self.generate\_random\_binary\_matrix(m=4, N=8, seed=42)

        self.extract\_mode\_standard = tk.BooleanVar(value=True)

        self.extract\_mode\_hash = tk.BooleanVar(value=False)

        self.colors = {

            "primary": "#4a6fa5",

            "primary\_dark": "#3d5d8a",

            "secondary": "#6c757d",

            "bg": "#f8f9fa",

            "text": "#212529",

            "light\_accent": "#e9ecef",

            "success": "#28a745",

            "warning": "#ffc107"

        }

        self.style = ttk.Style()

        self.style.theme\_use('clam')

        self.configure\_style()

        self.setup\_ui()

    # --- Интерфейс - визуальная соостовляющая

    def configure\_style(self):

        self.style.configure('TButton', background=self.colors["primary"], foreground='white', font=('Segoe UI', 10), padding=6)

        self.style.map('TButton', background=[('active', self.colors["primary\_dark"])])

        self.style.configure('TLabel', background=self.colors["bg"], foreground=self.colors["text"], font=('Segoe UI', 10))

        self.style.configure('TEntry', fieldbackground=self.colors["light\_accent"], font=('Segoe UI', 10))

        self.style.configure('TFrame', background=self.colors["bg"])

        self.style.configure('Primary.TButton', background=self.colors["primary"], foreground='white')

        self.style.configure('Success.TButton', background=self.colors["success"], foreground='white')

        self.style.map('Success.TButton', background=[('active', '#218838')])

        self.style.configure('Secondary.TButton', background=self.colors["secondary"], foreground='white')

        self.style.map('Secondary.TButton', background=[('active', '#5a6268')])

    # --- Интерфейс - настройка кнопок/полей

    def setup\_ui(self):

        self.root.geometry("800x600")

        self.root.configure(bg=self.colors["bg"])

        main\_frame = ttk.Frame(self.root, padding="20 20 20 20", style='TFrame')

        main\_frame.pack(fill=tk.BOTH, expand=True)

        header\_frame = ttk.Frame(main\_frame, style='TFrame')

        header\_frame.pack(fill=tk.X, pady=(0, 20))

        title\_label = tk.Label(header\_frame, text="Steganography Tool", font=('Segoe UI', 16, 'bold'), bg=self.colors["bg"], fg=self.colors["primary"])

        title\_label.pack(side=tk.LEFT)

        # --- блок изображения

        file\_frame = ttk.LabelFrame(main\_frame, text="Image", padding="10 10 10 10")

        file\_frame.pack(fill=tk.X, pady=(0, 15))

        self.file\_status = tk.StringVar(value="Image not loaded")

        file\_status\_label = ttk.Label(file\_frame, textvariable=self.file\_status)

        file\_status\_label.pack(side=tk.LEFT, fill=tk.X, expand=True)

        load\_btn = ttk.Button(file\_frame, text="Load Image", command=self.load\_image, style='TButton')

        load\_btn.pack(side=tk.LEFT, padx=(10, 0))

        # --- блок текста

        message\_frame = ttk.LabelFrame(main\_frame, text="Message", padding="10 10 10 10")

        message\_frame.pack(fill=tk.BOTH, expand=True, pady=(0, 15))

        self.text\_status = tk.StringVar(value="Text not loaded")

        text\_status\_label = ttk.Label(message\_frame, textvariable=self.text\_status)

        text\_status\_label.pack(anchor=tk.W, pady=(0, 5))

        self.text\_display = tk.Text(message\_frame, height=6, wrap=tk.WORD, font=('Segoe UI', 10))

        self.text\_display.pack(fill=tk.BOTH, expand=True, pady=(0, 10))

        load\_text\_btn = ttk.Button(message\_frame, text="Load Text", command=self.load\_text, style='TButton')

        load\_text\_btn.pack(anchor=tk.E, padx=(0, 5))

        btn\_frame = ttk.Frame(message\_frame, style='TFrame')

        btn\_frame.pack(fill=tk.X)

        embed\_standard\_btn = ttk.Button(btn\_frame, text="Embed Message (Standard)", command=self.embed\_message\_standard, style='Success.TButton')

        embed\_standard\_btn.pack(side=tk.LEFT)

        embed\_hash\_btn = ttk.Button(btn\_frame, text="Embed Message (with Hash)", command=self.embed\_with\_hash, style='Primary.TButton')

        embed\_hash\_btn.pack(side=tk.LEFT, padx=(10, 0))

        extract\_btn = ttk.Button(btn\_frame, text="Extract Message", command=self.extract\_message, style='Secondary.TButton')

        extract\_btn.pack(side=tk.LEFT, padx=(10, 0))

        result\_frame = ttk.LabelFrame(main\_frame, text="Result", padding="10 10 10 10")

        result\_frame.pack(fill=tk.BOTH, expand=True)

        self.output\_text = tk.Text(result\_frame, wrap=tk.WORD, bg=self.colors["light\_accent"], relief=tk.FLAT, font=('Segoe UI', 10), height=10)

        # Кнопка очистки результата

        clear\_result\_btn = ttk.Button(result\_frame, text="Clear Result", command=lambda: self.output\_text.delete(1.0, tk.END), style='Danger.TButton')

        clear\_result\_btn.pack(anchor=tk.E, pady=(5, 0))

        self.output\_text.pack(fill=tk.BOTH, expand=True)

        self.output\_text.config(state='normal')

        self.output\_text.bind("<Control-c>", lambda e: self.root.clipboard\_append(self.output\_text.selection\_get()))

        # ---------------------------

        mode\_frame = ttk.Frame(message\_frame, style='TFrame')

        mode\_frame.pack(anchor=tk.W, pady=(10, 0))

        ttk.Label(mode\_frame, text="Extraction mode:").pack(side=tk.LEFT, padx=(0, 10))

        ttk.Checkbutton(

            mode\_frame, text="Standard", variable=self.extract\_mode\_standard

        ).pack(side=tk.LEFT, padx=(0, 10))

        ttk.Checkbutton(

            mode\_frame, text="With Hash", variable=self.extract\_mode\_hash

        ).pack(side=tk.LEFT)

    # --- изображение

    def load\_image(self):

        path = filedialog.askopenfilename(

            initialdir="src/img\_in",

            filetypes=[("Image files", "\*.png \*.bmp")])

        if path:

            try:

                image = Image.open(path).convert('L')

                image\_array = np.array(image)

                self.container\_image = image\_array.copy()

                self.stego\_image = image\_array.copy()  # сохраняем исходное изображение

                self.file\_status.set("Image loaded successfully.")

            except Exception as e:

                messagebox.showerror("Error", f"Failed to load image: {e}")

    # --- текст

    def load\_text(self):

        path = filedialog.askopenfilename(

            initialdir="src/texts",

            filetypes=[("Text files", "\*.txt")])

        if path:

            try:

                with open(path, 'r', encoding='utf-8') as file:

                    self.text\_input = file.read()

                    self.text\_display.delete(1.0, tk.END)  # очистить поле

                    self.text\_display.insert(tk.END, self.text\_input)  # вставить текст

                    self.text\_status.set("Text loaded successfully.")

            except Exception as e:

                messagebox.showerror("Error", f"Failed to load text: {e}")

    def save\_image(self, array, nameOutImg):

        # 4. Сохраняем изображение

        output\_dir = "src/img\_out/"

        if not os.path.exists(output\_dir):

            os.makedirs(output\_dir)

        output\_dir = os.path.join(output\_dir, nameOutImg+".bmp")

        image = Image.fromarray(array.astype(np.uint8))

        image.save(output\_dir)

        self.output\_text.insert(tk.END,f"Message successfully embedded.\nSaved to {output\_dir}"+'\n')

    # --- Битовые утилиты ---

    def str\_to\_bits(self, s):

        return [int(b) for c in s for b in format(ord(c), '08b')]

    def bits\_to\_str(self, bits):

        chars = []

        for i in range(0, len(bits), 8):

            byte = bits[i:i+8]

            if len(byte) < 8:

                break

            chars.append(chr(int(''.join(map(str, byte)), 2)))

        return ''.join(chars)

    def xor\_bits(self, bits, key):

        return [b ^ key[i % len(key)] for i, b in enumerate(bits)]

    def get\_full\_bits(self, message\_bits):

        length\_bits = [int(b) for b in format(len(message\_bits), '032b')]

        return length\_bits + message\_bits

    def extracted\_bits\_message(self, extracted\_bits):

        extracted\_len = int("".join(map(str, extracted\_bits[:32])), 2)

        print(f"Extracted length: {extracted\_len}")

        return extracted\_bits[32:32 + extracted\_len]

    # --- Хэш-функция (линейная) ---

    def linear\_hash\_matrix(self, block, A):

        x = np.array(block, dtype=int)

        result = (A @ x) % 2

        return result.tolist()

    # --- Линейная хэш-функция: lambda(x) = A \* x mod 2 ---

    def generate\_random\_binary\_matrix(self, m, N, seed=None):

        if seed is not None:

            np.random.seed(seed)

        return np.random.randint(0, 2, size=(m, N), dtype=int)

    def add\_matrix\_hash\_blocks(self, bits, block\_size=8, A=None):

        assert A is not None, "Матрица A обязательна"

        m = A.shape[0]

        hashed = []

        for i in range(0, len(bits), block\_size):

            block = bits[i:i+block\_size]

            if len(block) < block\_size:

                break

            h = self.linear\_hash\_matrix(block, A)

            hashed.extend(block + h)

        return hashed

    def check\_and\_extract\_matrix\_hash\_blocks(self, bits, block\_size=8, A=None):

        assert A is not None, "Матрица A обязательна"

        m = A.shape[0]

        recovered = []

        for i in range(0, len(bits), block\_size + m):

            block = bits[i:i+block\_size]

            h = bits[i+block\_size:i+block\_size+m]

            if len(block) < block\_size or len(h) < m:

                break

            expected\_h = self.linear\_hash\_matrix(block, A)

            if h == expected\_h:

                recovered.extend(block)

        return recovered

    # --- Встраивание интерполяцией

    def interpolation\_method(self, img\_array, full\_message\_bits):

        height, width = img\_array.shape

        stego\_array = img\_array.copy()

        idx = 0

        total\_bits = len(full\_message\_bits)

        for i in range(height):

            for j in range(0, width - 1, 2):

                if idx >= total\_bits:

                    return stego\_array

                p1 = int(stego\_array[i, j])

                p2 = int(stego\_array[i, j + 1])

                mid = (p1 + p2) // 2

                desired\_bit = full\_message\_bits[idx]

                current\_bit = mid % 2

                if current\_bit != desired\_bit:

                    if mid % 2 == 0:

                        mid += 1

                    else:

                        mid -= 1

                    candidates = []

                    for delta1 in range(-4, 5):

                        for delta2 in range(-4, 5):

                            np1 = p1 + delta1

                            np2 = p2 + delta2

                            if 0 <= np1 <= 255 and 0 <= np2 <= 255 and (np1 + np2) // 2 == mid:

                                diff = abs(delta1) + abs(delta2)

                                candidates.append((diff, np1, np2))

                    if candidates:

                        \_, new\_p1, new\_p2 = min(candidates)

                        stego\_array[i, j] = new\_p1

                        stego\_array[i, j + 1] = new\_p2

                idx += 1

        return stego\_array

    def embed\_message\_standard(self):

        if self.stego\_image is None:

            messagebox.showerror("Error", "Please load an image first.")

            return

        message = self.text\_input

        if not message:

            messagebox.showerror("Error", "Please enter a message.")

            return

        message\_bits = self.str\_to\_bits(message)

        full\_bits = self.get\_full\_bits(message\_bits)

        self.stego\_image = self.interpolation\_method(self.container\_image, full\_bits)

        # --- save and notify

        self.save\_image(self.stego\_image, "standard\_embed")

    def embed\_with\_hash(self):

        if self.stego\_image is None:

            messagebox.showerror("Error", "Please load an image first.")

            return

        message = self.text\_input

        if not message:

            messagebox.showerror("Error", "Please enter a message.")

            return

        message\_bits = self.str\_to\_bits(message)

        encrypted\_bits = self.xor\_bits(message\_bits, self.key)

        hashed\_bits = self.add\_matrix\_hash\_blocks(encrypted\_bits, 8, self.A)

        full\_bits\_hashed = self.get\_full\_bits(hashed\_bits)

        self.stego\_image = self.interpolation\_method(self.container\_image, full\_bits\_hashed)

        # --- save and notify

        self.save\_image(self.stego\_image, "with\_hash\_embed")

    # --- Извлечение сообщения

    def extract\_message(self):

        if self.stego\_image is None:

            messagebox.showerror("Error", "No stego image available.")

            return

        use\_standard = self.extract\_mode\_standard.get()

        use\_hash = self.extract\_mode\_hash.get()

        if not (use\_standard or use\_hash):

            messagebox.showerror("Error", "Please select at least one extraction mode.")

            return

        extracted\_bits = self.extract\_bits\_form\_stego(self.stego\_image)

        if use\_standard:

            self.extract\_standard(extracted\_bits)

        else:

            self.extract\_with\_hash(extracted\_bits, self.key)

    def extract\_bits\_form\_stego(self, stego\_array):

        height, width = stego\_array.shape

        bits = []

        for i in range(height):

            for j in range(0, width - 1, 2):

                p1 = int(stego\_array[i, j])

                p2 = int(stego\_array[i, j + 1])

                interp = (p1 + p2) // 2

                bits.append(interp % 2)

        return bits

    def extract\_standard(self, extracted\_bits):

        extrcted\_bit = self.extracted\_bits\_message(extracted\_bits)

        self.output\_text.insert(tk.END, '\n📊 Максимально получилось встроить: '+ str(len(extrcted\_bit))+'\n')

        extracted\_msg = self.bits\_to\_str(extrcted\_bit)

        print("len extract message: ", len(extracted\_msg))

        self.output\_text.insert(tk.END, '\nExtrcted message: '+extracted\_msg+'\n')

    def extract\_with\_hash(self, extracted\_bits, key):

        raw\_encrypted = self.extracted\_bits\_message(extracted\_bits)

        recovered\_encrypted = self.check\_and\_extract\_matrix\_hash\_blocks(raw\_encrypted, 8, self.A)

        decrypted = self.xor\_bits(recovered\_encrypted, key)

        self.output\_text.insert(tk.END, '\n📊 Максимально получилось встроить: '+ str(len(decrypted))+'\n')

        extracted\_msg = self.bits\_to\_str(decrypted)

        print("len extract message: ", len(extracted\_msg))

        self.output\_text.insert(tk.END, 'Extrcted message: '+extracted\_msg+'\n')

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

    root = tk.Tk()

    app = SteganographyApp(root)

    root.mainloop()