Steganography Tool for Secure Text and File Hiding in Images

Introduction

In today's digital landscape, the need for discreet and secure communication has never been more critical. This project aims to build a comprehensive Steganography Tool that hides sensitive text messages (and optionally files) within digital images. Unlike encryption alone, steganography conceals the *existence* of the data, making it an ideal method for covert communication. The tool is equipped with a user-friendly graphical interface and supports both encrypted and non-encrypted message embedding.

Abstract

The Steganography Tool leverages image-based LSB (Least Significant Bit) techniques to embed messages into common image formats such as PNG and JPEG (auto-converted). The embedded message can be optionally encrypted using a secure key derived from a user-supplied password via the PBKDF2-HMAC-SHA256 method and Fernet encryption. The application is implemented using Python with GUI capabilities provided by Tkinter and TkinterDnD2 for drag-and-drop functionality. The system ensures automatic image format handling, visual feedback on image details, and secure data extraction. The tool also includes auto-renaming of output files and integrity verification of stegged content.

Tools Used

- **Programming Language**: Python 3.11
- **GUI Framework**: Tkinter, TkinterDnD2
- **Image Processing**: PIL (Pillow), NumPy
- **Encryption**: cryptography (Fernet, PBKDF2)
- **Testing & Validation**: Custom CLI script (V2_test_steganography.py)
- **Packaging**: PyInstaller (for Windows .exe builds)

Steps Involved in Building the Project

1. ### Design Phase
   The project was structured with clear modular separation — gui for interface, stegano for core logic, and tests for validation scripts.

2. ### GUI Development
   The Tkinter-based interface was built with intuitive components such as drag-and-drop support, “Add Image” button, message input area, encryption toggle (later nested within logic), and output status bar.

3. ### Image Preprocessing
   JPEG images were automatically converted to PNG before embedding to prevent lossy compression from corrupting hidden data. The image’s size and dimensions were shown dynamically.

4. ### Embedding Mechanism
   The message was embedded into the LSB of image pixels. If the user opted for encryption, the message was encrypted using a Fernet key derived from the password. A TXT:: or ENC:: prefix was added to help with automatic detection during extraction.

5. ### Extraction Mechanism
   During extraction, the tool first checks for prefixes. If ENC:: is detected, the user is prompted for a decryption password. Successful extraction automatically displays the hidden content.

6. ### Error Handling and Integrity Checks
   Edge cases such as missing images, corrupted data, or incorrect passwords were gracefully handled. Visual notifications were implemented throughout.

7. ### Testing & Packaging
   CLI test scripts were written to validate embedding/extraction for both encrypted and plaintext cases. The app was packaged into a standalone .exe for Windows.

Conclusion

The Steganography Tool successfully integrates LSB-based hiding with optional encryption into an accessible Python GUI. It enables secure and discreet communication through digital images with built-in integrity verification and error resilience. Its modular design, format handling, and user prompts make it both robust and easy to use. Future improvements could include multi-file embedding, steganalysis resistance, and cloud-based secure sharing.