Internship

Project Report

On

StegImage – LSB Steganography Tool

Submitted by:

ANDE SIREESHA

CSE – CYBER SECURITY

1. Introduction

With the rapid growth of digital communication, security and privacy have become critical concerns. Cryptography protects data confidentiality, but it is often noticeable that encryption has been applied. Steganography, on the other hand, hides the existence of communication by concealing information within harmless-looking media files such as images, audio, or video.

This project, **StegImage**, is a Python-based utility that implements **LSB (Least Significant Bit) image steganography** to embed text or files into digital images. The tool supports both **command-line interface (CLI)** and **graphical user interface (GUI)** modes, making it user-friendly for technical and non-technical users.

2. Objectives

The main objectives of this project are:

- To design and implement a tool for **hiding and extracting data** in images using LSB steganography.
- To support **text and file embedding** with or without password protection.
- To provide **AES encryption** for strong security, with an XOR fallback when AES is unavailable.
- To implement a **capacity checker** to estimate how much data can be hidden in a given image.
- To build an optional **GUI interface** using Tkinter with drag-and-drop support

3. Literature Survey

- Steganography in Digital Images
  LSB substitution is one of the simplest and most widely used methods for image steganography. It replaces the least significant bit of image pixels with the bits of the secret message. Since changes in the LSBs are visually indistinguishable, the hidden message remains invisible to the human eye.
- **Encryption in Steganography**
  Steganography can be combined with cryptography to provide an additional security layer. Even if the stego-image is intercepted, the payload remains encrypted. AES (Advanced Encryption Standard) is considered secure, while XOR encryption provides only lightweight obfuscation.
- **Existing Tools**
  Several tools such as **OpenStego** and **Steghide** exist, but they are often limited to command-line usage or require complex installation. StegImage focuses on being lightweight, cross-platform, and offering both CLI and GUI modes.

4. System Design

4.1 Architecture

The system is divided into four main modules:

1. **Payload Preparation**
   - Input text/file is taken from the user.
   - If a password is provided, the payload is encrypted using AES-CBC or XOR fallback.
2. **Container Construction**
   - A header is created with metadata (magic value, version, flags, payload size, optional filename).

- Payload (encrypted or plain) is appended.

3. **Embedding into Image**
    - Cover image pixels are read.
    - Payload bits are embedded into the least significant bits of RGB channels.
    - Output is saved as the stego-image.
4. **Extraction from Image**
    - Stego image is parsed to read the header.
    - Payload is extracted and decrypted (if encrypted).
    - Result is saved as text or file.

4.2 Technology Stack

- **Programming Language**: Python
- **Libraries**:
    - Pillow → Image processing
    - PyCryptodome (optional) → AES encryption
    - Tkinter & tkinterdnd2 (optional) → GUI interface
- **Algorithms**:
    - LSB substitution for steganography
    - AES-CBC for encryption
    - XOR fallback for lightweight obfuscation

5. Features

- Embed text messages into images.
- Embed any file into images.
- Extract hidden text or files from images.
- Password-protected embedding and extraction.

- AES encryption (recommended) with XOR fallback.
- Image capacity check before embedding.
- CLI for scripting and automation.
- GUI with drag-and-drop for ease of use.
- Built-in test suite to verify embedding/extraction.

6. Security Considerations

- **AES encryption** is highly secure and should always be preferred.
- **XOR encryption fallback** is not secure and should only be used when AES is unavailable.
- Stego-images should not be compressed (e.g., converting PNG → JPEG), as compression may destroy the hidden payload.
- Large payloads require large cover images; otherwise, the embedding process will fail.

7. Limitations

- Vulnerable to **statistical steganalysis** techniques.
- Not resistant to **lossy compression** (JPEG).
- Payload capacity is limited by the size of the cover image.
- XOR fallback is insecure for sensitive data.

8. Applications

- Secure message transmission.
- Hiding confidential files in plain sight.
- Digital watermarking.
- Educational demonstrations of cybersecurity concepts.

9. Conclusion

The **StegImage tool** successfully demonstrates how LSB steganography can be used to embed and extract hidden data in digital images. With AES encryption, it ensures strong security for sensitive data. Its hybrid support for CLI and GUI makes it both script-friendly and user-friendly.

Future improvements may include:

- Support for audio and video carriers.
- More advanced embedding techniques resistant to steganalysis.
- Integration with modern encryption schemes like ChaCha20.

10. References

1. Johnson, N. F., & Jajodia, S. (1998). Exploring steganography: Seeing the unseen. *Computer*, 31(2), 26-34.
2. Provos, N., & Honeyman, P. (2003). Hide and seek: An introduction to steganography. *IEEE Security & Privacy*, 1(3), 32-44.
3. Official Python Documentation – https://docs.python.org/
4. Pillow (Python Imaging Library) – https://python-pillow.org/
5. PyCryptodome Library – https://www.pycryptodome.org/