Project: Comprehensive Steganography System for Defense Applications

Team: The Encryptors
Hackathon: PYdroid
Date: July 26, 2025

1. Abstract

This document outlines the "Comprehensive Steganography System," an innovative project developed for secure information hiding within various digital media. Focusing on applications relevant to defense industries, this system integrates robust cryptographic techniques with advanced steganographic methods. It provides a secure, multi-media platform for covert data transmission, secure communication, and digital watermarking. The system features AES-256 GCM encryption for confidentiality and integrity, combined with Least Significant Bit (LSB) steganography across images, audio (WAV), and video (MP4/AVI) files.

2. Introduction

In an era where digital communication is ubiquitous, the need for secure and covert information exchange is paramount, especially in sensitive domains like defense. Steganography, the art and science of hiding information in plain sight, offers a powerful means to achieve this by embedding secret messages within innocuous cover media. Unlike cryptography, which scrambles data to make it unintelligible, steganography aims to conceal the very existence of the communication.

This project addresses the challenge of secure information hiding by developing a comprehensive steganographic system. It combines state-of-the-art encryption to protect the hidden data with sophisticated LSB techniques adapted for multiple media types. The goal is to provide a practical demonstration of how such a system can be implemented to enhance digital security and facilitate covert operations, ensuring that sensitive information remains undetected by unauthorized parties.

3. Core Technologies Used

The project leverages a diverse set of Python libraries to achieve its functionality:

- **Pillow (PIL):** The Python Imaging Library, essential for opening, manipulating, and saving image files (PNG, BMP, JPG, JPEG) for image steganography.
- **NumPy:** Fundamental package for numerical computing in Python, extensively used for efficient array operations on image pixel data, audio samples, and video frames.

- **OpenCV (cv2):** (Open Source Computer Vision Library) Used for reading, processing, and writing video files frame by frame, enabling video steganography.
- **cryptography:** A powerful and modern cryptographic library that provides robust implementations of cryptographic primitives, specifically used for AES-256 GCM encryption and key derivation (PBKDF2HMAC).
- **scipy.io.wavfile:** Part of SciPy, used for reading and writing WAV audio files, providing direct access to audio samples for LSB manipulation.
- **pathlib:** Python's object-oriented filesystem paths, used for robust and cross-platform handling of file and directory paths.
- **pydub:** A high-level audio manipulation library, useful for converting between different audio formats (e.g., MP3 to WAV) if needed, though direct LSB is performed on WAV.
- **soundfile:** A library for reading and writing sound files, providing a more comprehensive interface than wavfile for various audio formats.
- **librosa:** A Python library for music and audio analysis, included for potential future audio feature extraction or analysis, though not directly used in the core LSB embedding/extraction.
- **moviepy:** A library for video editing, used for more complex video manipulations if needed, though cv2 handles the core frame processing for LSB.
- **tkinter:** Python's standard GUI toolkit, used for the desktop application version.
- **pycryptodome:** A self-contained cryptographic library (imported as requested, but cryptography is used for core AES implementation for consistency with existing code).

4. System Architecture

The system is designed with a modular architecture, separating the core steganography logic from the user interface and file system interactions.

- **User Interface (Tkinter / Command-Line):**
    - Provides the primary means for user interaction. This can be a simple command-line interface (CLI) or a graphical user interface (GUI) built with Tkinter.
    - Allows users to input secret messages (text or file), specify cover media (image, audio, video), set passwords, and trigger steganography operations.
    - Displays success/error messages and output paths for stego files.
- **Core Steganography Logic (steganography_system.py - main script):**
    - Encapsulates all the heavy-lifting:
        - **Cryptography:** AES-256 GCM encryption/decryption, PBKDF2HMAC for key derivation.

        - **Data Conversion:** bytes_to_bits, bits_to_bytes.
        - **Media-Specific Helpers:** Functions for reading/writing WAV audio (scipy.io.wavfile), and processing video frames (cv2).
        - **StegoSystem Class:** The main orchestrator class that manages the entire process of generating/loading keys, encrypting/decrypting payloads, and performing media-specific LSB embedding/extraction.
- **File System Interaction:**
    - The system directly interacts with the local file system for:
        - Reading original cover media files from designated input directories.
        - Saving generated stego files to specified output directories.
        - Storing and retrieving AES encryption keys and salts.

```
+----------------+
|    User's      |
|  Interaction   |
| (Tkinter GUI / |
| Command Line)  |
+----------------+
       |
       | Calls Functions/Methods
       |
       v
+----------------------+
| Core Steganography    |
|       Logic          |
| (steganography_system.py) |
+-----------+-----------+
            |
            | Reads/Writes Files
            v
+------------------------+
|   File System Storage   |
| (Images/, Audio/, Videos/,|
|  output_stego_images/,  |
|  output_stego_audio/,   |
|  output_stego_videos/,  |
|  keys/)                 |
+------------------------+
```

5. Key Features & Functionality

5.1. Encryption (AES-256 GCM)

- **Algorithm:** Advanced Encryption Standard (AES) with 256-bit key size in Galois/Counter Mode (GCM).
- **Confidentiality & Integrity:** GCM provides both authenticated encryption (ensuring data hasn't been tampered with) and confidentiality.
- **Key Derivation:** Uses PBKDF2HMAC with SHA256 and 100,000 iterations to derive a strong 256-bit AES key from a user-provided password and a randomly generated salt.
- **Key Management:** AES keys and salts are saved to and loaded from binary files (.bin) in the keys/ directory, allowing for persistent key management.

5.2. Steganography (Least Significant Bit - LSB)

- **Method:** LSB steganography is employed, which involves altering the least significant bit(s) of the pixel/sample data in the cover medium. This method is chosen for its simplicity and minimal impact on visual/auditory perception.
- **Payload Structure:** The encrypted secret message (including nonce and authentication tag from GCM) is converted into a stream of bits. A 32-bit header representing the length of the *payload bits* is prepended to the actual payload, enabling accurate extraction.
- **Bit Manipulation:** Direct bitwise operations (& 0xFE, | bit) are used to embed and extract bits.

5.3. Supported Media Types

- **Images:**
  - **Formats:** PNG, BMP, JPG, JPEG (internally converted to RGBA for consistency).
  - **Embedding:** Bits are embedded into the LSB of the R, G, and B channels of each pixel.
  - **Capacity:** Each pixel can hold up to 3 bits (1 bit per R, G, B channel).
- **Audio:**
  - **Formats:** WAV (uncompressed PCM audio).
  - **Embedding:** Bits are embedded into the LSB of each audio sample across all channels.
  - **Capacity:** Each audio sample can hold 1 bit.
- **Video:**
  - **Formats:** MP4, AVI, MOV, MKV (processed frame by frame).
  - **Embedding:** Bits are embedded into the LSB of the B, G, and R channels of

each pixel within each video frame.
    - **Capacity:** Each pixel in each frame can hold up to 3 bits.

6. Implementation Details

6.1. Data Flow (Hide Operation)

1. **Secret Message Input:** User provides text or specifies a file path.
2. **Encoding:** Text is UTF-8 encoded; file content is read as raw bytes.
3. **Encryption:** The StegoSystem's encrypt_payload method uses AESCipher to encrypt the secret bytes with AES-256 GCM. The output is nonce + ciphertext + tag.
4. **Bit Conversion:** The encrypted payload (bytes) is converted into a list of individual bits (0s and 1s) using bytes_to_bits.
5. **Length Header:** The length of the payload_bits list (in bits) is converted to 4 bytes, then to 32 bits, and prepended to the payload_bits. This header is crucial for extraction.
6. **Cover Media Loading:** The chosen cover media (image, WAV audio, or video frames) is loaded into a NumPy array.
7. **Capacity Check:** The total available LSB capacity of the cover media is calculated and compared against the full_payload_bits length. If the message is too large, an error is returned.
8. **LSB Embedding:**
    - The cover media's pixel/sample data is flattened.
    - Each bit from full_payload_bits is sequentially embedded into the LSB of the corresponding pixel component/audio sample.
    - This is done by clearing the LSB (e.g., value & 0xFE) and then setting it to the secret bit (| bit).
9. **Stego Media Saving:** The modified NumPy array is converted back to the appropriate media format and saved as a new stego file.

6.2. Data Flow (Extract Operation)

1. **Stego Media Loading:** The stego media file is loaded into a NumPy array.
2. **Length Header Extraction:** The first 32 bits are extracted from the LSBs of the stego media. These bits are converted back to bytes, then to an integer, revealing the length of the *original encrypted payload bits*.
3. **Payload Extraction:** Using the extracted length, the remaining bits of the encrypted payload are extracted from the LSBs of the stego media, starting immediately after the length header.
4. **Byte Conversion:** The extracted payload bits are converted back into bytes using bits_to_bytes.

5. **Decryption:** The StegoSystem's decrypt_payload method uses AESCipher to decrypt the extracted bytes. This process verifies the GCM tag. If the tag is invalid (indicating tampering or incorrect key), decryption fails.
6. **Decoding:** The decrypted plaintext bytes are decoded back into a UTF-8 string (or raw bytes if the secret was a file).

6.3. Key Management

- **generate_aes_key(password, key_file, salt_file):** Derives a new AES key and salt from a password and saves them to specified files.
- **load_aes_key(password, key_file, salt_file):** Loads the salt from file, re-derives the AES key using the provided password and loaded salt, and initializes the AESCipher. This ensures the correct key is used and provides a basic form of password-based key management.

7. Setup & Running the Project

7.1. Prerequisites

- **Python 3.8+:** Download from python.org.
- **pip:** Python package installer (comes with Python).
- **ffmpeg:** (Highly Recommended for Video) A powerful multimedia framework. Download and add to your system's PATH. This is often required by opencv-python and moviepy for video encoding/decoding.
    - Download: ffmpeg.org/download.html

7.2. Project Structure

Ensure your project directory (PYdroid--Hack--project) has the following structure:

```
PYdroid--Hack--project/
├── steganography_system.py     # The main Python script (contains all logic and
GUI)
├── requirements.txt          # List of Python dependencies
├── Images/                   # Store your cover image files here (e.g., .png, .jpg)
├── Audio/                    # Store your cover WAV audio files here (e.g., .wav)
├── Videos/                   # Store your cover video files here (e.g., .mp4, .avi)
├── output_stego_images/       # Output directory for stego images (created
automatically)
├── output_stego_audio/        # Output directory for stego audio (created
automatically)
├── output_stego_videos/       # Output directory for stego videos (created
automatically)
```

```
├── keys/                # Directory for AES key and salt files (created automatically)
└── venv/                # (Optional but Recommended) Python Virtual Environment
```

7.3. Cloning the Repository (if applicable)

If this project is hosted on Git, clone it:

```
git clone https://github.com/devp-with-V/PYdroid--Hack--project.git
cd PYdroid--Hack--project
```

Otherwise, ensure you have all files in the correct structure.

7.4. Setting Up the Virtual Environment (Recommended)

It's highly recommended to use a virtual environment to manage project dependencies and avoid conflicts with other Python projects.

1. **Navigate to your project directory:**
   ```
   cd PYdroid--Hack--project
   ```
2. **Create the virtual environment:**
   ```
   python -m venv venv
   ```
3. **Activate the virtual environment:**
   - **On Windows (Git Bash/MinGW, PowerShell, Cmd):**
     ```
     ./venv/Scripts/activate
     ```
   - **On macOS/Linux:**
     ```
     source venv/bin/activate
     ```

Your command prompt should now show (venv) at the beginning.

7.5. Installing Dependencies

With the virtual environment activated, install all required libraries using requirements.txt:

```
pip install -r requirements.txt
```

7.6. Running the System

After all dependencies are installed and your virtual environment is active, run the main Python script:

```
python steganography_system.py
```

This will launch the Tkinter graphical user interface.

8. Usage Guide (Desktop Application)

1. **Launch the Application:** Run python steganography_system.py from your activated virtual environment. A desktop window will appear.
2. **AES Setup Tab:**
   - **Crucial First Step:** Before performing any steganography operations, you **must** set up your AES key.
   - Enter a strong password in both "AES Password" and "Confirm Password" fields.
   - Click "Generate & Save AES Key" to create and store the key/salt files in the keys/ folder. The "AES Key Status" will update to "Loaded".
   - Alternatively, if you've already generated keys, you can enter the password and click "Load AES Key" to load existing keys from the keys/ folder.
3. **Image Steganography Tab:**
   - **Secret Message:** Enter text directly in the text field or click "Browse" to select a file (e.g., .txt, .pdf, .bin) containing your secret message.
   - **Cover Image:** Click "Browse" to select a PNG, BMP, JPG, or JPEG image file from your system.
   - **Output Stego Image Name:** Provide a name for the resulting stego image file (e.g., my_stego_image.png). This file will be saved in the output_stego_images/ folder.
   - **Encrypt & Hide:** Click this button to embed the encrypted secret message into the chosen cover image. A success/error message will pop up, and details will be logged in the "System Log" at the bottom.
   - **Extract & Decrypt:** To extract a hidden message, click "Browse" under "Stego Image (for Extraction)" to select the stego image file. Ensure your AES key is loaded with the correct password, then click "Extract & Decrypt". The extracted (and decrypted) message will appear in the "Extracted Message" text area.
4. **Audio Steganography Tab:**
   - Similar to Image Steganography.
   - **Secret Message:** Enter text or browse for a secret file.

    - **Cover Audio:** Click "Browse" to select a **WAV** (.wav) audio file. (Note: MP3s are not directly supported for LSB; convert them to WAV first using external tools if needed).
    - **Output Stego Audio Name:** Provide a name (e.g., my_stego_audio.wav). This file will be saved in the output_stego_audio/ folder.
    - **Encrypt & Hide / Extract & Decrypt:** Follow the same process as for images.
5. **Video Steganography Tab:**
    - Similar to Image/Audio Steganography.
    - **Secret Message:** Enter text or browse for a secret file.
    - **Cover Video:** Click "Browse" to select an MP4, AVI, or MOV video file.
    - **Output Stego Video Name:** Provide a name (e.g., my_stego_video.mp4). This file will be saved in the output_stego_videos/ folder.
    - **Encrypt & Hide / Extract & Decrypt:** Follow the same process. Note that video processing can take significant time depending on file size and system performance.
6. **About Tab:**
    - Provides information about the system, its features, and disclaimers.
7. **System Log:**
    - The text area at the bottom of the application window displays real-time messages about operations, successes, and errors, providing detailed feedback.

9. Security Considerations

- **Encryption Strength:** AES-256 GCM provides strong confidentiality and integrity, making it extremely difficult for unauthorized parties to read or tamper with the hidden message without the correct AES key. PBKDF2HMAC ensures the key derived from your password is robust against brute-force attacks.
- **LSB Limitations:** While LSB steganography is simple and effective for many covert communication scenarios, it is generally considered **fragile** against certain types of attacks:
    - **Steganalysis:** Advanced statistical analysis tools can sometimes detect the presence of LSB-embedded data.
    - **Lossy Compression:** Re-compressing a stego file (e.g., re-saving a JPEG, converting WAV to MP3, or re-encoding a video) will likely destroy the hidden LSB data. The system outputs to formats that are generally LSB-friendly (PNG, WAV, MP4 with specific codecs) but users should be aware of this limitation.
- **Key Management:** The security of the system heavily relies on the strength of

the user's password and the secure storage of the derived AES keys and salts.

10. Future Enhancements

- **Advanced Steganography Algorithms:** Implement more robust steganography techniques like Discrete Cosine Transform (DCT) based methods for images/videos (more resilient to compression) or phase encoding for audio.
- **Multiple Embedding Layers:** Allow embedding multiple layers of data or using different LSB depths (e.g., 2-bit LSB) if perceptual quality allows.
- **Performance Optimization:** Implement multi-threading or GPU acceleration for faster video processing.
- **Error Reporting:** More detailed error logging and user feedback.
- **Progress Indicators:** Implement real-time progress bars for long-running operations (especially video).
- **Cross-Platform Compatibility:** Further testing and optimization for various operating systems.

11. Conclusion

The Comprehensive Steganography System successfully demonstrates a robust approach to secure information hiding across images, audio, and video. By combining strong AES-256 GCM encryption with practical LSB steganography and a user-friendly desktop interface, the project provides a functional prototype for covert communication. While LSB has inherent fragilities, the added layer of strong encryption ensures the confidentiality and integrity of the hidden message, making it a valuable tool for educational and research purposes in the field of digital security.