Steganography

Team members:

1. Ali Mohamed
2. Omar Ahmed
3. Marios Maged
4. Hagar Ali
5. Hana Ahmed

Contents

[Introduction: 3](#_Toc198571384)

[**Codebase Structure:** 4](#_Toc198571385)

[**Implemented Steganography Methods** 4](#_Toc198571386)

[**Image Steganography** 4](#_Toc198571387)

[**Audio Steganography** 5](#_Toc198571388)

[**Video Steganography** 6](#_Toc198571389)

[**Security Features** 6](#_Toc198571390)

[**Technical Implementation Analysis** 7](#_Toc198571391)

[**Cryptography Module** 7](#_Toc198571392)

[**Image Steganography Implementation** 9](#_Toc198571393)

[**LSB Implementation Analysis** 9](#_Toc198571394)

[**DCT Implementation Analysis** 12](#_Toc198571395)

[**Audio Steganography Implementation** 17](#_Toc198571396)

[**LSB Implementation Analysis** 17](#_Toc198571397)

[**Echo Implementation Analysis** 20](#_Toc198571398)

[**Video Steganography Implementation** 25](#_Toc198571399)

[**LSB Implementation Analysis** 25](#_Toc198571400)

[**Command Line Interface** 32](#_Toc198571401)

[**Conclusion** 34](#_Toc198571402)

[**References** 34](#_Toc198571403)

# Introduction:

Steganography is the art and science of hiding information within other, seemingly innocuous data, in such a way that the presence of the hidden information is undetectable. Unlike cryptography, which protects the content of a message by making it unreadable to unauthorized users, steganography conceals the very existence of the message itself. This makes it a powerful technique for secure communication, especially in environments where overt encryption might raise suspicion.

Historically, steganography has been used in various forms—ranging from invisible inks and microdots to subtle alterations in text. With the advancement of digital technologies, modern steganography has evolved to embed information within digital media such as images, audio files, videos, and even network protocols. Techniques often involve manipulating the least significant bits (LSBs) of multimedia files, which allows data to be hidden without noticeably altering the original content.

In our experiment, we developed a collection of Python modules that apply various forms of steganography, including techniques for hiding data in images, audio, and video files. The system supports multiple embedding and extraction methods, such as Least Significant Bit (LSB), Echo Hiding, and Discrete Cosine Transform (DCT).

# **Codebase Structure:**

The codebase is organized into the following main components:

1. **Cryptography Module** (crypto.py)  
   Provides encryption and binary conversion utilities used across all steganography methods.
2. **Image Steganography** (image\_stego.py)  
   Implements methods for embedding messages in image files.
3. **Audio Steganography** (audio\_stego.py)  
   Implements methods for embedding messages in audio files.
4. **Video Steganography** (video\_stego.py)  
   Implements methods for embedding messages in video files.
5. **Command Line Interface** (main.py)  
   Provides user-friendly access to the steganography functionality.

# **Implemented Steganography Methods**

## **Image Steganography**

1. **Least Significant Bit (LSB) Method**

* **Function**: hide\_message\_in\_image\_lsb() / extract\_message\_from\_image\_lsb()
* **Description**: Modifies the least significant bit of RGB pixel values to store binary message data.
* **Characteristics**: High capacity, relatively low visual impact, but vulnerable to statistical analysis.

1. **Discrete Cosine Transform (DCT) Method**

* **Function**: hide\_message\_in\_image\_dct() / extract\_message\_from\_image\_dct()
* **Description**: Embeds message in the frequency domain by modifying DCT coefficients.
* **Characteristics**: More resistant to compression and some image processing operations than LSB.

## **Audio Steganography**

1. **Least Significant Bit (LSB) Method**

* **Function**: hide\_message\_in\_audio\_lsb() / extract\_message\_from\_audio\_lsb()
* **Description**: Similar to image LSB, modifies the least significant bits of audio samples.
* **Characteristics**: High capacity but vulnerable to statistical analysis.

1. **Echo Hiding Method**

* **Function**: hide\_message\_in\_audio\_echo() / extract\_message\_from\_audio\_echo()
* **Description**: Embeds data by introducing slight echoes with different delay times to represent binary data.
* **Characteristics**: More resistant to statistical attacks but with lower capacity than LSB.

## **Video Steganography**

1. **Frame-based LSB Method**

* **Function**: hide\_message\_in\_video() / extract\_message\_from\_video()
* **Description**: Distributes the encrypted message across multiple frames using LSB embedding.
* **Characteristics**: Preserves original audio track and uses delimiters for message metadata.

## **Security Features**

1. **Encryption**

* All messages are encrypted before embedding using XOR encryption with a password-derived key (SHA-256 hash).
* Provides a layer of security even if steganographic hiding is detected.

1. **Delimiter Usage**

* Each steganographic method implements delimiters to mark the end of hidden messages.
* Common pattern: appending 1111111111111110 to binary message data.

1. **Password Protection**

* All operations require a password for both hiding and extraction.

# **Technical Implementation Analysis**

## **Cryptography Module**

The crypto.py module provides four core functions:

1. text\_to\_binary(): Converts text to binary representation.

A computer screen shot of text

AI-generated content may be incorrect.

1. binary\_to\_text(): Converts binary back to text with appropriate padding.

A computer screen shot of a program

AI-generated content may be incorrect.

1. generate\_key\_from\_password():Derives a secure 32-byte encryption key from a password and salt using PBKDF2 with SHA-256, encoded in URL-safe base64 format.

A screen shot of a computer

AI-generated content may be incorrect.

1. encrypt(): Encrypts a message using a password-derived key with Fernet encryption, embedding a random salt to ensure secure, unique ciphertext output each time.

A screen shot of a computer

AI-generated content may be incorrect.

1. decrypt(): Decrypts a base64-encoded message using a password-derived key and the embedded salt, returning the original plaintext or an error message if decryption fails.

A computer screen shot of a program

AI-generated content may be incorrect.

## **Image Steganography Implementation**

The image steganography module implements two distinct approaches to hiding data within images: Least Significant Bit (LSB) modification and Discrete Cosine Transform (DCT) coefficient manipulation. Both methods offer different trade-offs between capacity, robustness, and detectability.

### **LSB Implementation Analysis**

The hide\_message\_in\_image\_lsb() function takes the following approach:

1. **Preprocessing**:
   * Encrypts the message using the provided password
   * Converts the encrypted message to binary form
   * Appends the delimiter '1111111111111110' to mark the end of the message
2. **Image Preparation**:
   * Opens the image using PIL and converts it to RGB mode if necessary
   * Calculates maximum message capacity: (width \* height \* 3) // 8 bytes
   * Validates that the message can fit within the image
3. **Embedding Process**:
   * Iterates through each pixel, modifying the least significant bit of each RGB channel:
   * Each pixel can store 3 bits (one in each RGB channel)
   * Embedding stops once the entire message is hidden
4. **Output**:
   * Saves the modified image as PNG (lossless format to preserve LSB modifications)
5. **Code**:

A computer screen shot of a program

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

The extract\_message\_from\_image\_lsb() function reverses this process:

1. **Extraction**:
   * Reads LSB from each RGB channel of each pixel
   * Concatenates bits to form the binary message
   * Monitors for the delimiter pattern to identify message end
2. **Postprocessing**:
   * Removes the delimiter
   * Applies padding if needed
   * Converts binary back to encrypted text
   * Decrypts the message using the provided password
3. **Code**:

A screen shot of a computer program

AI-generated content may be incorrect.

### **DCT Implementation Analysis**

The hide\_message\_in\_image\_dct() function implements a more sophisticated approach:

1. **Preprocessing**:
   * Encrypts and converts the message to binary form with delimiter
   * Reads the image using OpenCV
   * Converts from BGR to YCrCb color space (separating luminance from chrominance)
2. **DCT Processing**:

* Divides the Y channel (luminance) into 8×8 pixel blocks
* For each block where a message bit needs to be stored:
  + - Applies 2D Discrete Cosine Transform
    - Modifies a specific DCT coefficient (position [4,5]) based on bit value:
    - Applies inverse DCT to return to spatial domain

1. **Key Optimization**:

* Uses the mid-frequency DCT coefficient at position [4,5]
* Allows strength parameter to control embedding intensity (default: 25.0)
* Clips values to ensure they remain in valid range (0-255)

1. **Output**:

* Converts back to BGR color space
* Saves as high-quality JPEG (quality=100)

1. **Code**:

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

The extract\_message\_from\_image\_dct() function performs extraction by:

1. **DCT Analysis**:
   * Converting to YCrCb color space
   * Processing each 8×8 block with DCT
   * Checking if the coefficient at [4,5] is above the threshold
   * Building binary message based on coefficient signs
2. **Message Recovery**:
   * Detecting delimiter to find message end
   * Converting binary to encrypted text
   * Decrypting with provided password

3. **Code**:

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

## **Audio Steganography Implementation**

### **LSB Implementation Analysis**

The hide\_message\_in\_audio\_lsb() function takes the following approach:

1. **Preprocessing**

* Encrypts the message using the provided password
* Converts the encrypted message to binary form
* Appends the delimiter '1111111111111110' to mark the end of the message

1. **Audio Preparation**

* Opens the audio file using the wave module
* Retrieves audio parameters: channels, sample width, frame rate, number of frames
* Calculates maximum message capacity based on available audio samples
* Validates that the message can fit within the audio file

1. **Embedding Process**

* Iterates through audio samples, modifying the least significant bit of each sample:
  + Clears the LSB of each sample with bitwise AND operation (~1)
  + Sets the LSB according to the message bits with bitwise OR operation
  + Each sample stores 1 bit of the message
* Embedding continues until the entire message (including delimiter) is hidden

1. **Output**

* Writes the modified audio data to the output file, preserving all original audio parameters
* Maintains the same sample width, frame rate, and number of channels

1. **Code**

A computer screen shot of a program code

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

The extract\_message\_from\_audio\_lsb() function performs extraction by:

1. **LSB Extraction**

* Reads the steganographic audio file
* Extracts the least significant bit from each audio sample
* Concatenates the bits to form a binary message
* Locates the delimiter ('1111111111111110') to identify the end of the message
* Converts the binary data back to text and decrypts using the provided password

1. **Code**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen with text

AI-generated content may be incorrect.**

### **Echo Implementation Analysis**

The hide\_message\_in\_audio\_echo() function takes the following approach:

**1.Preprocessing**

* Encrypts the message using the provided password
* Converts the encrypted message to binary form
* Appends the delimiter '1111111111111110' to mark the end of the message

**2.Audio Preparation**

* Reads the audio file using scipy.io.wavfile
* Normalizes audio data to float32 format for processing
* Divides the audio into segments of 0.1 seconds each
* Calculates maximum message capacity based on number of segments
* Validates that the message can fit within the available segments

**3.Embedding Process**

* Iterates through each segment, encoding binary data using echo patterns:
  + Uses a short delay (1ms) to represent bit '0'
  + Uses a longer delay (3ms) to represent bit '1'
  + Applies a decay factor (0.6) to the echo signal
  + Adds the echo to the original segment
  + Normalizes the output to prevent clipping
* Each 0.1-second segment encodes exactly 1 bit of information

**4.Output**

* Converts the modified audio back to the original format (e.g., int16)
* Preserves stereo channels if present in the original file
* Writes the modified audio to the output file

**5.code**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

The extract\_message\_from\_audio\_echo () function performs extraction by:

**1. Echo Extraction**

* **Reads the steganographic audio file**
* **Divides the audio into 0.1-second segments**
* **For each segment:** 
  + **Applies a Hamming window to reduce spectral leakage**
  + **Computes the cepstrum (FFT → log → IFFT) to detect echo patterns**
  + **Examines the cepstrum at specific delay positions with tolerance**
  + **Determines the bit value by comparing cepstral magnitudes**
* **Concatenates the extracted bits and locates the delimiter**
* **Converts the binary data to text and decrypts using the provided password**

**2.code**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

## **Video Steganography Implementation**

### **LSB Implementation Analysis**

The hide\_message\_in\_audio\_lsb() function takes the following approach:

**1. Preprocessing**

* **Encrypts the message using the provided password**
* **Normalizes the encrypted message to ASCII format (removing non-ASCII characters)**
* **Splits the message into individual characters for frame-by-frame embedding**
* **Creates metadata with delimiters to facilitate extraction:** 
  + **First delimiter: "^$^" (marks the beginning of metadata)**
  + **Word delimiter: "^\*^" (separates characters during embedding)**
  + **Frame delimiter: "^#^" (marks the end of the message)**

**2. Video Preparation**

* **Opens the source video file using OpenCV**
* **Validates video parameters (dimensions, frame rate)**
* **Calculates required frames based on message length:** 
  + **Starting from frame #10 (configurable starting point)**
  + **Each character requires one frame for embedding**
* **Verifies the video has sufficient frames to accommodate the message**
* **Initializes output video with lossless FFV1 codec to preserve data integrity**

**3. Embedding Process**

* **Performs two distinct embedding operations:** 
  + **Embeds metadata in the first frame (including start frame position)**
  + **Embeds message characters in sequential frames starting from frame #10**
* **For each frame where data is embedded:** 
  + **Converts character data to 7-bit binary representation**
  + **Modifies the least significant bit (LSB) of RGB channels in selected pixels**
  + **Uses a systematic pattern (every 4th pixel is skipped)**
  + **Each pixel stores 3 bits (one in each RGB channel)**
* **Embedding continues until all characters (with appropriate delimiters) are hidden**

**4. Output Processing**

* **Preserves the original audio by:** 
  + **Extracting audio from the original video**
  + **Creating temporary audio file**
  + **Applying the audio to the steganographic video**
* **Produces final output using lossless video codec (FFV1) and AAC audio codec**
* **Cleans up temporary files (audio and intermediate video files)**

**5.code**

**A computer screen with many colorful text

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

The extract\_message\_from\_audio\_lsb() function performs extraction by:

**1. LSB Extraction**

* **Reads the steganographic video frame by frame**
* **From the first frame:** 
  + **Extracts metadata containing the start frame position**
  + **Validates the presence of required delimiters**
* **From subsequent frames (beginning at the identified start frame):** 
  + **Extracts LSB values from RGB channels of pixels**
  + **Converts binary data to characters**
  + **Accumulates characters until frame delimiter is found**
* **Assembles the complete encrypted message**
* **Decrypts the message using the provided password**

**2.code**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

# **Command Line Interface**

The main application provides a user-friendly command-line interface with the following commands:

1. **hide: Embeds a message in a specified file**

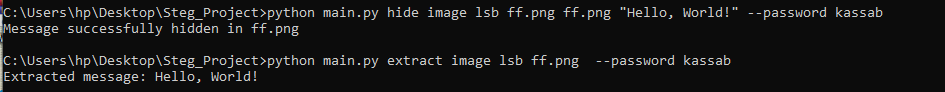
python main.py hide [image/audio/video] [method] [input\_file] [output\_file] [message] --password [password]

1. **extract: Extracts a message from a specified file**

python main.py extract [image/audio/video] [method] [input\_file] --password [password]

Sample for running a code:

Image with LSB:



Audio with LSB:

A black background with white text

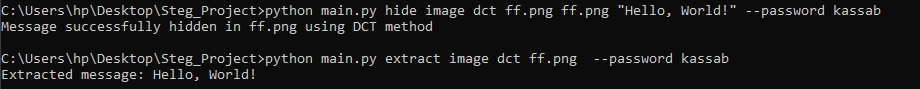
AI-generated content may be incorrect.

Video with LSB:

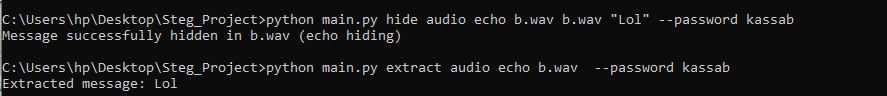
A screen shot of a computer screen

AI-generated content may be incorrect.

Image with DCT:



audio with echo:



# **Conclusion**

The analyzed steganography codebase provides a comprehensive toolkit for hiding and extracting messages across various media types. It demonstrates good software engineering practices with its modular design, error handling, and command-line interface. While the implemented steganographic methods are effective for casual use, they may not withstand sophisticated steganalysis techniques.

For educational and non-critical applications, the codebase serves as an excellent introduction to the principles and implementation of digital steganography. For security-critical applications, additional hardening against steganalysis and stronger encryption would be recommended.

# **References**

1. <https://taylorandfrancis.com/knowledge/Engineeringandtechnology/Computer_science/Steganography/>

2. <https://ieeexplore.ieee.org/document/9335027>

3. <https://www.geeksforgeeks.org/what-is-steganography/>

4. <https://www.edureka.co/blog/steganography-tutorial>

5. <https://medium.com/better-programming/a-guide-to-video-steganography-using-python-4f010b32a5b7>