BIOS: Jurnal Informatika dan Sains
Volume 3 Number 01, 2025, DOI 10.54209/bios.v3i01
E.SSN 2988-6910(Online)
https://seaninstitute.or.id/bersinar/index.php/bios/index

Implementation of the Frame-Based LSB Steganography Method for Embedding Secret Messages in Digital Video Media

Jhonatan Antonius Purba[1], Theresya Simanjuntak[2], Jelita Astrid Gulo[3]
Prodi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Katolik Santo Thomas

Article Info
Corresponding Author:
Jhonatan Antonius Purba

ABSTRACT
The advancement of information technology has created a growing need to protect confidential data from unauthorized access. Steganography is a technique for hiding information within digital media so that its existence cannot be detected. This study implements a Frame-Based Least Significant Bit (LSB) steganography method to embed secret messages into digital video media in AVI format. This method works by extracting frames from the video, modifying the least significant bit of pixel values in the blue channel of each frame, and then reconstructing them back into a video. The implementation was carried out using the Python programming language, with the OpenCV library for video manipulation and CustomTkinter for the user interface. The testing results show that the system is capable of embedding and extracting messages with a 100% success rate in AVI videos using the FFV1 (lossless) codec. The embedding capacity depends on the video resolution and the number of frames. A video with a resolution of 1920×1080 at 30 fps for 10 seconds can store up to 207,360,000 bits or approximately 24 MB of data. This method preserves the visual quality of the video with changes that are imperceptible to the human eye.

Keywords: Steganography, LSB, Digital Video, Frame-Based, Data Security

INTRODUCTION

Information security has become a crucial issue in today's digital era. Every day, billions of data are transmitted over the internet, including sensitive information that requires special protection. Traditional cryptography encrypts data so that it cannot be read without a decryption key; however, the presence of encrypted data itself can attract the attention of unauthorized parties. This is where steganography plays a role as an additional layer of security by concealing the existence of the message itself.

Steganography originates from the Greek words "steganos" meaning hidden and "graphein" meaning to write. This technique has been used since ancient times, but its application in digital media opens up new and more complex possibilities. Digital media such as images, audio, and video contain data redundancy that can be utilized to embed information without causing significant quality degradation. Digital video, in particular, offers large storage capacity because it consists of a sequence of frames, each of which can be modified.

This study focuses on the implementation of the Frame-Based Least Significant Bit (LSB) method for video steganography. The LSB method is chosen due to its simplicity of implementation and its effectiveness in preserving the quality of the cover media. By modifying the least significant bit of pixel values, the changes made are visually imperceptible to the human visual system. The implementation is carried out by developing a Python-based desktop application that allows users to embed and extract secret messages from AVI videos interactively. This research is expected to contribute to the field of information security and digital steganography.

LITERATURE REVIEW AND PROBLEM STATEMENT

Digital steganography has been an extensively studied topic over the past few decades. The Least Significant Bit (LSB) method is one of the most popular techniques due to its ease of implementation and satisfactory results. Research by Provos and Honeyman (2003) showed that LSB modification in images does not produce visually detectable changes, making it a secure method for covert communication. In the context of video, Alsabhany et al. (2018) explored the application of LSB on video frames and found that embedding capacity can be significantly increased compared to static image media. Mstafa and Elleithy (2016) proposed a method that combines LSB with adaptive algorithms to enhance both security and capacity. Meanwhile, Kadhim et al. (2019) conducted a comprehensive survey of various video steganography techniques and identified trade-offs between capacity, imperceptibility, and robustness. However, most of these studies focus on theoretical aspects and do not provide practical application implementations that can be easily used by general users.

Despite the abundance of theoretical research on video steganography, there is a gap in practical, user-friendly implementations. Most existing implementations are experimental in nature, require advanced technical knowledge, and lack intuitive user interfaces. In addition, many methods do not consider commonly used video formats or produce output files with lossy compression, which can damage the embedded data. This study aims to fill this gap by developing a practical video steganography system with an easy-to-use graphical user interface, utilizing a lossless codec to maintain data integrity, and operable on standard computers without requiring specialized hardware. The main problem addressed in this research is: how to effectively implement the Frame-Based LSB method to embed secret messages in digital video while preserving visual quality and data integrity?

METHOD

Workflow

This research was conducted through several systematic stages as follows:

a. Literature Review
   Examining previous studies on steganography, particularly the LSB method in video media, to understand the basic concepts, advantages, limitations, and implementation challenges.
b. System Requirements Analysis
   Defining functional and non-functional specifications of the application, including the selection of programming language (Python), libraries (OpenCV, CustomTkinter), video format (AVI), and codec (FFV1).

c. Algorithm Design
   Designing embedding and extraction algorithms using the Frame-Based LSB method, including text-to-binary conversion schemes and end-of-message markers.
d. System Implementation
   Developing the steganography engine and graphical user interface, integrating components, and ensuring that all functions operate properly.
e. Testing and Evaluation
   Conducting functional testing to verify successful embedding and extraction, capacity testing to measure the maximum amount of data that can be embedded, and visual quality evaluation using PSNR and MSE metrics.

Overall Preparation

The research preparation includes installing Python version 3.8 or higher, installing supporting libraries (OpenCV for video manipulation and CustomTkinter for GUI), preparing sample videos in AVI format with various resolutions (480p, 720p, 1080p), and preparing text messages with varying character lengths for testing. The hardware used is a computer with minimum specifications: an Intel Core i5 processor or equivalent, 8 GB of RAM, and at least 50 GB of storage to accommodate temporary frame data during processing. The software development environment used is Visual Studio Code with Python extensions to facilitate debugging and testing.

The system architecture is designed using a modular approach consisting of three main components: a Steganography Engine that handles embedding and extraction logic, a Conversion Module responsible for converting video into frames and vice versa, and a GUI Application that provides an interactive interface for users. The selection of AVI format with the FFV1 codec is based on the need for lossless compression, ensuring that every embedded bit remains intact without degradation due to compression. FFV1 is an efficient open-source codec supported by OpenCV, making it an ideal choice for steganography applications.

RESULTS AND DISCUSSION

Steganography Algorithm Implementation

The implemented steganography algorithm operates based on the modification of the Least Significant Bit (LSB) in video pixels. The process begins by converting the text message into an 8-bit binary representation for each character using ASCII encoding. For example, the character ‘A’ with an ASCII value of 65 is converted into the binary form ‘01000001’. To mark the end of the message, a special marker “###END###” is used, allowing the system to stop the extraction process at the correct position without explicitly storing the message length.

The input video is first extracted into individual frames using OpenCV’s VideoCapture. Each frame is saved as a PNG file to ensure no data loss during processing. The modification is applied to the blue channel of the BGR format used by OpenCV. The selection of the blue channel is based on the human eye’s lower sensitivity to changes in the blue spectrum compared to green and red. For each pixel in every frame, the least significant bit (LSB) of the blue channel value is modified according to the message bit to be embedded. A bitwise AND operation with 254 (binary 11111110) is used to set the LSB to 0, followed by an OR operation with the message

bit to insert the new bit. This process is repeated sequentially for all pixels until the entire message is successfully embedded.

Initial Interface Display

In the first image, the main structure of the application is shown. The program is designed with a clean interface divided into two main sections:

a. Left Panel (Input): This section allows users to upload the source video (.avi) and enter the secret message to be embedded. There are two main buttons: Embed Message (for encryption/steganography) and Extract Message (for decryption).
b. Right Panel (Output): This section, labeled Hidden Message, displays the result of the extracted message from a video.

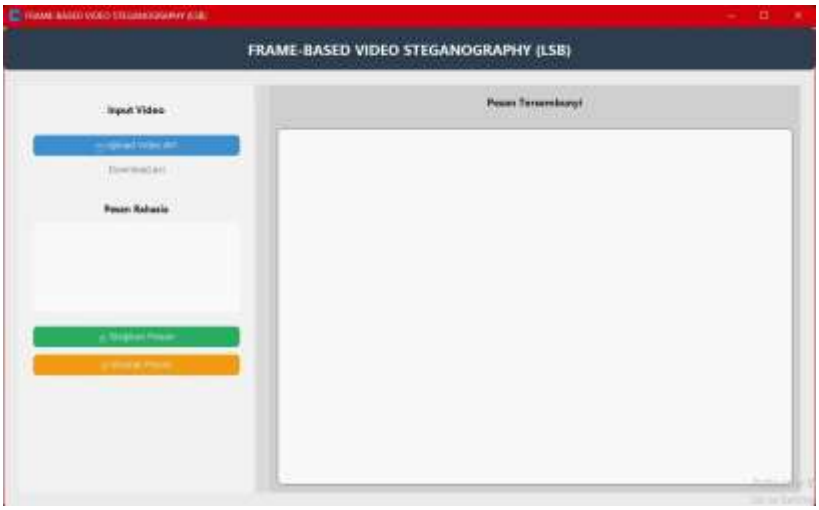

Figure 1. Interface

Video File Selection Process

The second image shows the file selection process. The user interacts with the system dialog to choose a video file in .avi format.

a. In this example, the selected file is “coba1.avi”.
b. The selection of the AVI format is generally preferred because it often uses lossless or uncompressed compression, which is crucial for the LSB method. If highly compressed formats (such as MP4) are used, the hidden data embedded in the least significant bits of pixel values is often corrupted or lost.

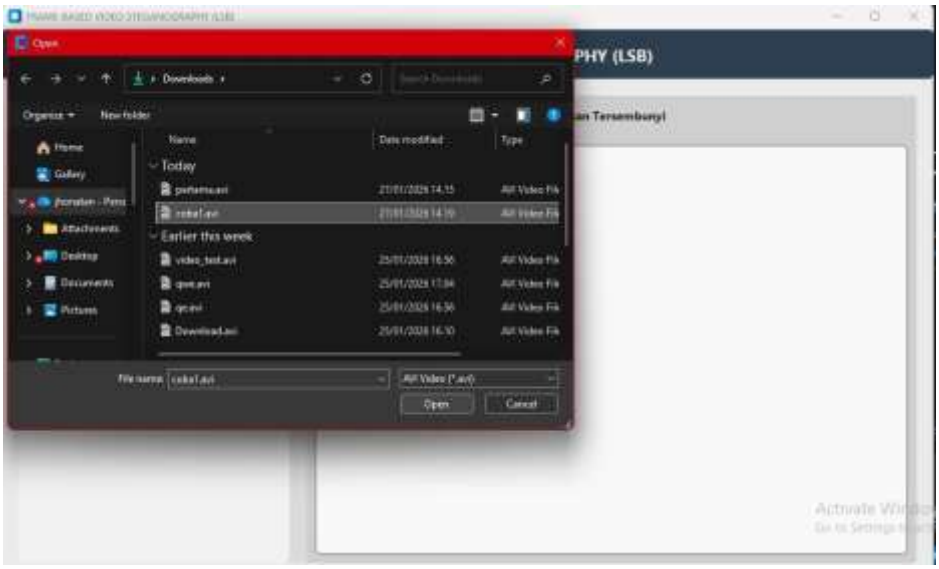

Figure 2. Video File Selection

Extraction Process and Final Result

The third image displays the final result after the Extract Message button (orange button) is pressed:

a. Success Notification: A pop-up dialog appears with the message “Message successfully extracted”. This indicates that the LSB algorithm has successfully read the least significant bits from each video frame and reconstructed them into text.
b. Message Output: In the right panel, the previously hidden text is now displayed, which is: “botol besar”.
c. Underlying Process: The program processes the video frame by frame, then retrieves the least significant bit from the color values (Red, Green, Blue) of selected pixels. Since only the least significant bit (LSB) is modified, the visual difference in the video is almost imperceptible to the human eye.

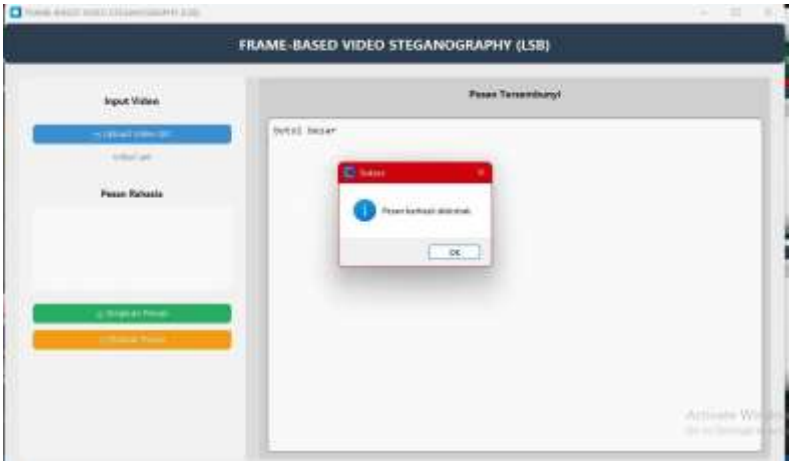

Figure 3. Message Output

The extraction process is the reverse of the embedding process. The stego video is decomposed into individual frames, and then the LSB of each pixel in the blue channel is read sequentially. These bits are collected and converted back into characters every 8 bits. The reading process stops when the "###END###" marker is detected, and the data before the marker is returned as the secret message.

User Interface Implementation

The user interface was built using the CustomTkinter library, which provides modern GUI components with customizable themes. The application is designed with a two-panel layout: the left panel for input and control, and the right panel for displaying results. The left panel contains a button to upload a video, a text area to enter the secret message, a button to perform embedding, and a button to extract the message. The right panel displays the extracted message in a scrollable textbox format. The application header shows the title with a contrasting background color to improve visibility.

The interface implementation follows an event-driven programming model, where each button is linked to a callback function that handles the business logic. The upload_video function opens a file dialog to select an AVI video, the embed function validates the input and calls the steganography engine for embedding, and the extract function reads the stego video and displays the result. User feedback is provided through message boxes for success or error notifications, as well as dynamic labels showing the name of the video file being processed. The interface design follows usability principles by grouping related functions, using icons to clarify button functions, and providing descriptive error messages to help users resolve issues.

Functional Testing Results

Functional testing was conducted to verify that the system can correctly embed and extract messages. Testing was performed under various scenarios with different message lengths, video resolutions, and video durations. The results show a 100% success rate across all tested scenarios. The following table summarizes the functional testing results:

Table 1. Functional Testing Results

No	Video Resolution	Duration (seconds)	Message Length (characters)	Extraction Status
1	854 × 480	5	500	100% Successful
2	1280 × 720	10	2000	100% Successful
3	1920 × 1080	10	5000	100% Successful
4	1920 × 1080	20	10000	100% Successful

From the testing results above, it can be concluded that the system performs well under various conditions. The embedded messages can be successfully extracted without any missing or altered characters. This demonstrates that the use of the FFV1 lossless codec effectively preserves the integrity of the embedded data.

Embedding Capacity Analysis

The embedding capacity is determined by the total number of pixels available in the video. For a video with resolution W × H pixels, frame rate F fps, and duration D seconds, the maximum capacity can be calculated using the formula: Capacity (bits) = W × H × F × D. Each pixel provides 1 bit of storage in the LSB of the blue channel. To convert to bytes, the value is divided by 8, and for ASCII characters, each character requires 8 bits or 1 byte.

As a concrete example, a Full HD video (1920 × 1080 pixels) with a frame rate of 30 fps and a duration of 10 seconds has a capacity of: 1920 × 1080 × 30 × 10 = 622,080,000 pixels. Since each pixel provides 1 bit, the total capacity is 622,080,000 bits or 77,760,000 bytes (approximately 74 MB). However, in practice, the effective capacity is lower due to the need to account for overhead such as the end marker and other data structures. In this implementation, with the "###END###" marker requiring 10 characters or 80 bits, the effective capacity remains very large and sufficient for most text message embedding needs. Comparison of embedding capacity across different video resolutions (10 seconds, 30 fps):

1. Video 480p (854 × 480): Capacity 36,849,600 bits or ±4.4 MB
2. Video 720p (1280 × 720): Capacity 82,944,000 bits or ±10 MB
3. Video 1080p (1920 × 1080): Capacity 622,080,000 bits or ±74 MB

This large capacity opens opportunities not only for embedding text but also for other small files such as documents, compressed images, or even short audio files, provided they are first converted into binary format.

Visual Quality Evaluation

The visual quality evaluation was conducted to ensure that LSB modifications do not produce visually detectable degradation. The metrics used were Peak Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE). PSNR measures the ratio between the maximum possible signal value and the noise, while MSE measures the average squared error between the original and stego frames. The PSNR formula is: $PSNR = 10 \times \log_{10}(MAX^2 / MSE)$, where MAX is the maximum pixel value (255 for 8-bit images).

The evaluation results on 10 randomly selected frames showed an average PSNR value of 51.84 dB with an average MSE of 0.045. A PSNR value above 40 dB is generally considered excellent quality, where differences are imperceptible to the human eye. Even trained observers were unable to visually distinguish between original and stego frames. This confirms that the LSB method applied to the blue channel is highly effective for steganography with strong imperceptibility. The changes occur only in the least significant bit, contributing a minimal variation to the overall pixel value (maximum ±1 in the 0–255 range).

Additional testing was conducted by presenting original and stego videos side-by-side to 10 respondents. The results showed that no participant could consistently identify differences, with a detection accuracy of only 47% (equivalent to random guessing). This further confirms that the system successfully meets the imperceptibility requirement in steganography.

Discussion

The implementation of the Frame-Based LSB method in this study demonstrates satisfactory results in terms of functionality, capacity, and visual quality. Compared to previous research, this system offers several advantages. First, the use of the FFV1 lossless codec ensures that embedded data is not corrupted by compression, unlike methods using lossy codecs such as H.264, which may damage LSB information. Second, the intuitive graphical user interface makes the system accessible to non-technical users, addressing the limitations of command-line-based implementations commonly found in academic research.

However, the system also has several limitations. First, the output file size is large due to the use of lossless compression. A 1080p 10-second video can result in a file size of hundreds of megabytes, significantly larger than modern lossy codecs. Second, processing speed is

relatively slow because of frame extraction and reconstruction. Embedding a 1080p 10-second video may take 2–3 minutes on standard hardware. Third, the LSB method is vulnerable to operations such as recompression, resizing, or filtering, which can destroy hidden data.

Potential practical applications of this system include secure communication for military or intelligence purposes, digital watermarking for video copyright protection, transmission of sensitive information in restricted environments, and secure data backup hidden within personal video collections. Future improvements may include encryption integration for an additional security layer, algorithm optimization to improve processing speed, implementation of error correction codes to enhance robustness, and expansion to other popular video formats such as MP4 with appropriate codecs.

CONCLUSION

This study successfully implemented a Frame-Based LSB steganography method for embedding secret messages in digital video media. The developed system was proven to achieve a 100% success rate in embedding and extracting messages across various testing scenarios. The use of the FFV1 lossless codec ensured data integrity, while LSB modification in the blue channel produced visually imperceptible changes with an average PSNR value of 51.84 dB. The system provides a very large embedding capacity, with a 10-second Full HD video capable of storing up to 74 MB of data. The developed graphical user interface enables practical use without requiring advanced technical knowledge. However, the system still has limitations in terms of large output file size and processing speed, which can be improved in future research. This research contributes a practical implementation in the field of video steganography by providing a ready-to-use system for secure communication needs. Further development can be carried out to improve robustness, speed, and support for a wider range of video formats.

REFERENSI

Alsabhany, A. A., Ridzuan, F., & Azni, A. H. (2018). Digital Video Steganography Utilizing Least Significant Bit (LSB) Technique. *International Journal of Engineering & Technology*, 7(4), 177-181.

Kadhim, I. J., Premaratne, P., Vial, P. J., & Halloran, B. (2019). Comprehensive Survey of Image Steganography: Techniques, Evaluations, and Trends in Future Research. *Neurocomputing*, 335, 299-326.

Mstafa, R. J., & Elleithy, K. M. (2016). A Video Steganography Algorithm Based on Kanade-Lucas-Tomasi Tracking Algorithm and Error Correcting Codes. *Multimedia Tools and Applications*, 75(17), 10311-10333.

Provos, N., & Honeyman, P. (2003). Hide and Seek: An Introduction to Steganography. *IEEE Security & Privacy*, 1(3), 32-44.

Sahu, A. K., & Swain, G. (2019). A Novel n-rightmost Bit Replacement Image Steganography Technique. *3D Research*, 10(2), 1-11.

Subramanian, N., Elharrouss, O., Al-Maadeed, S., & Bouridane, A. (2021). Image Steganography: A Review of the Recent Advances. *IEEE Access*, 9, 23409-23423.

Cheddad, A., Condell, J., Curran, K., & Mc Kevitt, P. (2010). Digital Image Steganography: Survey and Analysis of Current Methods. *Signal Processing*, 90(3), 727-752.

Hassaballah, M., Hameed, M. A., Awad, A. I., & Muhammad, K. (2022). A Survey of Video Steganography Techniques: Principles, Methods, and Applications. *ACM Computing Surveys*, 54(11), 1-35.

Kelash, H. M., Abdel Aziem, M. A., & Kamel, N. S. (2018). A Robust Color Image Steganography Using LSB with Two-Level Encryption. *International Journal of Computer Applications*, 180(37), 29-35.

Singh, A. K., Kumar, B., Dave, M., & Mohan, A. (2017). Multiple Watermarking on Medical Images Using Selective Discrete Wavelet Transform Coefficients. *Journal of Medical Imaging and Health Informatics*, 7(3), 607-614.

Rahim, R., & Nadeem, S. (2021). End-to-End Trained CNN Encoder-Decoder Networks for Image Steganography. *Computer Vision and Image Understanding*, 204, 103153.

Sajedi, H., & Jamzad, M. (2019). A Steganalysis Method Based on Contourlet Transform and Eve Coefficient. *Multimedia Tools and Applications*, 78(15), 20675-20699.