International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307 OPEN ACCESS

Available Online at : www.ijsrcseit.com

doi : https://doi.org/10.32628/CSEIT2342434

Embedding and Extracting Hidden Messages in Images Using LSB-Based Steganography

Dr. G. Sreedhar

Professor, Department of Computer Science, National Sanskrit University, Tirupati, Andhra Pradesh, India

ARTICLEINFO

Article History:

Accepted: 01 April 2023

Published: 12 April 2023

Publication Issue

Volume 9, Issue 2

March-April-2023

Page Number

848-851

ABSTRACT

This paper explores the implementation of image-based steganography utilizing the Least Significant Bit (LSB) method to embed and extract hidden messages within digital images. The approach leverages the imperceptibility of minor pixel alterations to conceal information, ensuring the visual integrity of the cover image. The study provides a detailed explanation of the LSB technique, its implementation, and potential applications in secure communication.

Keywords : Steganography, Least Significant Bit (LSB), Image hiding, Secure communication, Data embedding

I. INTRODUCTION

Steganography, the art of concealing information within other non-suspicious data, has been employed for centuries to facilitate covert communication. With the advent of digital media, particularly images, steganography has found new avenues for application. The Least Significant Bit (LSB) method is one of the most widely used techniques due to its simplicity and effectiveness. This paper delves into the LSB method, demonstrating its application in embedding and extracting hidden messages within digital images.

II. BACKGROUND AND RELATED WORK

The LSB technique involves modifying the least significant bit of the pixel values in an image to embed secret information. This method is preferred for its simplicity and the minimal perceptible change it introduces to the image. However, the method's security is limited, as the embedded information can be extracted through statistical analysis and image processing techniques. Recent advancements have introduced more robust techniques, such as adaptive LSB and encrypted LSB methods, to enhance security and capacity.

III. METHODOLOGY

3.1 Encoding Process

The encoding process involves the following steps:

1. **Convert the Secret Message to Binary:**
   - Convert each character of the secret message to its ASCII value.
   - Convert each ASCII value to an 8-bit binary string.
   - Concatenate all the binary strings to form a single binary stream.
2. **Prepare the Cover Image:**
   - Obtain the pixel data of the cover image.
   - Ensure the image has sufficient pixels to accommodate the secret message.
   - Initialize a variable to keep track of the current bit position in the secret message.
3. **Embed the Secret Message:**
   - Iterate through each pixel of the cover image:
   - For each pixel, extract the Red, Green, and Blue (RGB) values.
   - For each color channel (R, G, B):
     - Retrieve the current bit from the secret message.
     - Set the least significant bit (LSB) of the color channel to the current bit.
     - Update the color channel with the modified value.
     - Move to the next bit in the secret message.
   - If all bits of the secret message have been embedded, terminate the process.
4. **Save the Stego Image:**
   - After embedding all bits, save the modified image as the stego image.

3.2 Decoding Process

The decoding process entails:

1. **Prepare the Stego Image:**
   - Obtain the pixel data of the stego image.
   - Initialize a variable to store the extracted binary stream.
2. **Extract the Secret Message:**
   - Iterate through each pixel of the stego image:
     - For each pixel, extract the Red, Green, and Blue (RGB) values.
     - For each color channel (R, G, B):
     - Retrieve the least significant bit (LSB) of the color channel.
     - Append the LSB to the extracted binary stream.
3. **Convert Binary to Text:**
   - Group the extracted binary stream into 8-bit segments.
   - Convert each 8-bit segment to its corresponding ASCII character.

- Concatenate all characters to form the secret message.

III. IMPLEMENTATION

The implementation utilizes HTML5 and JavaScript to create a client-side application for embedding and extracting hidden messages in images. The HTML5 <canvas> element is employed to manipulate image data, while JavaScript's FileReader API facilitates image loading. The encoding and decoding functions operate directly within the browser, ensuring user privacy and data security. The encoding and decoding process are implemented with a small web program.

1. Encoding Process (Embedding the Secret Message)

Input:

- coverImage: The original image to embed the message into.
- secretMessage: The text message to be concealed.

Output:

- stegoImage: The image with the embedded secret message.

```
function encodeMessage(coverImage, secretMessage):
    binaryMessage =
convertTextToBinary(secretMessage) + "00000000"  //
Append null byte to indicate end
    imageData = getImageData(coverImage)
```

```
pixelIndex = 0

for each bit in binaryMessage:
    pixel = imageData[pixelIndex]
    for each colorChannel in pixel:
        lsb = getLSB(colorChannel)
        newColorChannel = setLSB(colorChannel, bit)
        updatePixel(pixel, newColorChannel)
        pixelIndex += 1

stegoImage = createImageFromData(imageData)
return stegoImage
```

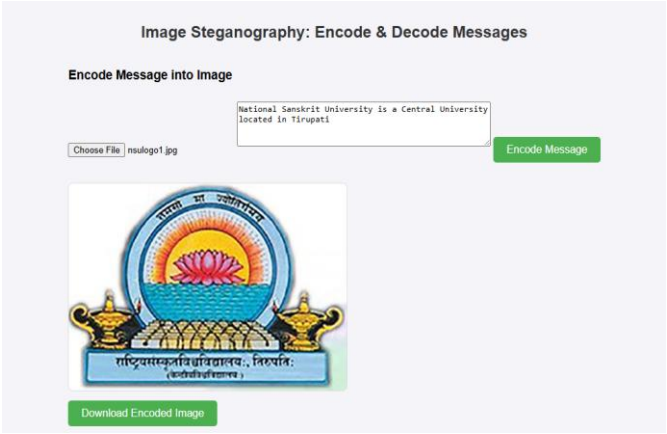

Figure 1: Encoding Process

2. Decoding Process (Extracting the Hidden Message)

Input:

- stegoImage: The image containing the hidden message.

Output:

- decodedMessage: The extracted secret message.

```
function decodeMessage(stegoImage):
    imageData = getImageData(stegoImage)
    binaryMessage = ""
    pixelIndex = 0

    while True:
        pixel = imageData[pixelIndex]
        for each colorChannel in pixel:
            lsb = getLSB(colorChannel)
            binaryMessage += lsb
            pixelIndex += 1
        if binaryMessage ends with "00000000":
            break
    decodedMessage =
            convertBinaryToText(binaryMessage)
    return decodedMessage
```

Helper Functions

```
function convertTextToBinary(text):
    binaryString = ""
    for each character in text:
        asciiValue = getASCII(character)
        binaryString += convertTo8BitBinary(asciiValue)

    return binaryString

function convertBinaryToText(binaryString):
    text = ""
    for i from 0 to length(binaryString) by 8:
        byte = binaryString[i:i+8]
        asciiValue = convertBinaryToDecimal(byte)
        character = getCharacterFromASCII(asciiValue)
        text += character
    return text

function getImageData(image):
    // Extract pixel data from the image
    return pixelData

function createImageFromData(pixelData):
    // Create an image from the pixel data
    return image

function getLSB(colorChannel):
    return colorChannel & 1  // Extract the least
significant bit

function setLSB(colorChannel, bit):
    if bit == 1:
```

```
        return colorChannel | 1  // Set the least
significant bit to 1
    else:
        return colorChannel & 0xFE  // Set the least
significant bit to 0

function updatePixel(pixel, newColorChannels):
    // Update the pixel with the new color channels
    pixel = newColorChannels
```

Figure 2: Decoding Process

IV. RESULTS AND DISCUSSION

The implemented tool successfully encodes and decodes messages within images, with the hidden information remaining imperceptible to the human eye. The approach demonstrates the feasibility of client-side steganography using standard web technologies. However, the method's security is limited by the simplicity of the LSB technique, which is vulnerable to statistical analysis and image processing attacks. Future enhancements could involve integrating more robust algorithms and employing encryption to bolster security.

V. CONCLUSION

This study illustrates the potential of HTML5 and JavaScript in implementing accessible, client-side steganographic solutions. While the current implementation serves as an introductory tool for educational purposes, it lays the groundwork for more sophisticated applications in secure communication and digital watermarking.

REFERENCES

[1]. Fridrich, J., & Goljan, M. (2002). Digital Image Steganography Using the LSB Matching Revisited. Proceedings of the 4th Information Hiding Workshop.

[2]. Bassil, Y. (2012). Image Steganography based on a Parameterized Canny Edge Detection Algorithm. arXiv:1212.6259.

[3]. Hashemi, S. H. O., Majidi, M. H., & Khorashadizadeh, S. (2022). Color Image Steganography using Deep Convolutional Autoencoders based on ResNet Architecture. arXiv:2211.09409.

[4]. Halboos, E. H. J., & Albakry, A. M. (2022). Hiding text using the least significant bit technique to improve cover image in the steganography system. Bulletin of Electrical Engineering and Informatics, 11(6), 4337–4344.

[5]. Rashid, A., Missen, M. M. S., & Salamat, N. (2015). Analysis of Steganography Techniques using Least Significant Bit in Grayscale Images and its Extension to Colour Images. Journal of Scientific Research and Reports, 9(3), 1–14.

[6]. Gupta, S., Goyal, A., & Bhushan, B. (2012). Information Hiding Using Least Significant Bit Steganography and Cryptography. International Journal of Modern Education and Computer Science, 4(6), 27–34.

[7]. Gangurde, S., & Tiwari, K. (2020). LSB Steganography Using Pixel Locator Sequence with AES. arXiv preprint arXiv:2012.02494.)

[8]. Abdulla, A. A., Sellahewa, H., & Jassim, S. A. (2020). Improving embedding efficiency for digital steganography by exploiting similarities between secret and cover images. arXiv preprint arXiv:2004.11974.

[9]. Lerch-Hostalot, D., & Megías, D. (2017). Unsupervised Steganalysis Based on Artificial Training Sets. arXiv preprint arXiv:1703.00796.

[10]. Khan, M., Ahmad, J., Farman, H., & Zubair, M. (2015). A Novel Image Steganographic Approach for Hiding Text in Color Images using HSI Color Model. arXiv preprint arXiv:1503.00388.