Course:
KH4017CEM Introduction to Programming

steganography application Course Work

Student Name: Yassin Mohamed Elprince

Student ID: CU202300182

Lecturer: Shaimaa M. Mohamed

Date of submission: 4th of December 2024

Table of contents

Contents

Introduction

Confidentiality, integrity, and authentication are three pillars of the information security world and are commonly known by the CIA. They are used for the protection of any communication between two or multiple entities. To achieve this protection, the information must only be accessible to authorized individuals or corporations which is the purpose of confidentiality. It can be achieved through security measures such as steganography. In some cases, messages cannot be sent as plain text and must be encrypted to avoid the risk of exposing valuable information from malicious individuals that can be positioned between the sender and the receiver. In this case, the Image Message Hider application is a solution made for steganography, which means hiding a secret message inside something, such as an image or a file. This application's main goal is to let users hide private information within an image file without changing how the picture looks. This can help in many situations to achieve safe communication and secure sensitive data.

What Is It Used For?

Safe communication: People can send hidden messages without making them obvious. This is useful for those wanting to share important information safely.

Data Security: The application helps put important data inside pictures, making it difficult for unauthorized members to access the information.

How it works:

The application goes through a series of steps to change the pixel data of an image and hide the message in it.

1. Convert Message to Binary: The first step is to convert the secret message into a binary form (0s and 1s). The input is changed into an 8-bit binary string for each character.
2. Image opening: The application opens an image file (jpeg, jpg) using the Pillow library which enables it to work with images.
3. Embed the Message: The application changes the least significant bit (LSB) of the color red channel of every pixel in the image. The LSB is the lowest order bit of a binary number, so modifying it would have a very low impact on the resulting color of the pixel, so it becomes invisible to the human eye.

Instructions for using the application.

To use The Image Message Hider application there are some instructions to follow.

1. Open the Image Message Hider application.
2. Paste the image file path where the message will be hidden.
3. Type the secret message you want to hide within the image.
4. Specify the output path for the modified image with the secret.
5. Enter the Hide Message button to embed the message into the image.

A table of unit tests.

Here is a table of unit tests that can be found in the unit test file stored on GitHub:

Test	*Method*	*Expected output*
A normal string “AB”	String to binary	0100000101000010
An Empty string “ ”	String to binary	an empty string “ ”
String with space “A B”	String to binary	010000010010000001000010
String with numbers “123”	String to binary	001100010011001000110011
String with multiple spaces “A B”	String to binary	0100000100100000001000000100001 0
Mixed string “A1!”	String to binary	010000010011000100100001
Image path “image.jpg”, secret message “hi”, output path “modified.jpg”	Hide message in image	Modified image (with the secret message “hi”)

Repository:

When we develop projects it's essential to store and manage our application code & files efficiently. GitHub (open source) is a distributed version control system that tracks changes, shares, and collaborates with local and remote team members. If this steganography code application is made publicly available on Git Hub, others can also use it. It makes it easy for users to download, modify, and contribute to the project.

Steganography code Here in the repository:

https://github.com/YassinElprince/steganography.git

Description of the algorithm

The code implements the least significant bit (LSB) steganography algorithm. This technique is commonly performed to conceal secret messages input into digital images by changing the least significant bits of the pixel values, so it won't be visible to the naked eye. Here's an in-depth look at the algorithm and what it gets to work on.

Steps of the Algorithm:

- The code begins by importing the pillow library to give access to image manipulation, to use this library, make sure to install it in the terminal before running the code by running the appropriate command (pip install Pillow).
- Converting the message (string) to binary, using the function (string_to_bin(message)), this function takes the input secret and converts every single character to a unique 8-bit binary string, and joins them together. It is later embedded into the image by first using an input secret message, which is the string that needs to be changed into binary. Then the function uses a generator expression to iterate over each character in the input string using (for char in message) then for each character, the function (ord) is called which gives the character an integer value, the integer value is then converted to an 8-bit binary string using the (format) function. The format ensures that the binary representation is always 8 bits long, filling with leading zeros if necessary. Finally, the (join) function joins all the binary strings generated for each character into a single string.
- The function (hide_message_in_image) embeds the secret message into an image file by using the least significant bit (LSB), it manipulates the pixels of the image to hide the message without changing the picture and won't be noticeable to the human eye. Firstly, the code receives the image path (image_path), then the secret message (secret_message), and finally the output path of the modified image (output_path). The function then opens the image using the pillow library and converts the message using the function (string_to_bin(secret_message)). It then retrieves the pixel data of the image and determines its width and height. Using (data_index), it keeps track of the bit of the binary message that is currently being embedded, using the nested loop to iterate over each pixel in the image. For each pixel, it checks if there are still bits left to embed (if data_index < data_length). If there are still bits left, they will be modified to the least significant bit of the red channel of the pixel to embed the current bit of the binary message. After embedding the entire message, the modified image is saved to the specified output path.
- Finally, the last and main function starts by printing a welcome message to the user. This informs the user about the purpose of the program, The function then asks the user to provide three pieces of information (image path, the secret message, and the modified image output path). After collecting the necessary inputs, the function calls (hide_message_in_image). This function is responsible for embedding the secret message into the specified image and saving the modified image to the given output path. Once the message has been successfully hidden in the image, the function prints a confirmation message to the console, indicating that the process is complete and specifying where the modified image has been saved. The function (if __name__ == "__main__":) ensures that the main part of the script only runs when the script is executed directly.

Conclusion

The steganography that forms the basis of the Image Message Hider application applies the Least Significant Bit (LSB) steganography technique to camouflage confidential messages in image files. Through the advanced image manipulation abilities of the Pillow library, this application does not allow the hidden data to be perceived by the naked eye, thereby preserving the expression of that image. String to binary conversions with precise pixel manipulation manifests the toughness of this algorithm for real world applications in secure communication and data protection. On the technology advancement front, the project reinforces a modern complement to cybersecurity needs, a sensitive information protection tool. Moreover, by publishing the application on GitHub, this project encourages collaborative evolution and invites open source contributions in steganography. It shall serve as a stepping stone for the Image Message Hider to reinforce secure communication in the future.

Source code:

```python
from PIL import Image

def string_to_bin(message):  # Convert each character in the message to ito binary
    return ''.join(format(ord(char), '08b') for char in message)

def hide_message_in_image(image_path, secret_message, output_path):
    image = Image.open(image_path)
    binary_message = string_to_bin(secret_message) + '1111111111111110'  # Convert the secret message to binary and append a delimiter to mark the end
    data_index = 0   # Index to track the current bit of the message being hidden
    data_length = len(binary_message) # Total length of the binary message
    pixels = image.load()   # Load the pixel data of the image
    width, height = image.size   # Get the dimensions of the image

    for y in range(height):   # go over each pixel in the image
        for x in range(width):
            if data_index < data_length:   # Check if there are still bits to hide
                r, g, b = pixels[x, y]    # Get the RGB values of the current pixel
                r = (r & ~1) | int(binary_message[data_index])   # Modify the LSB of the red channel to hide the message bit and Set LSB to the message bit
                pixels[x, y] = (r, g, b)    # Update the pixel with the modified red value
                data_index += 1  # Move to the next bit of the message

    image.save(output_path)

def main():
    print("Welcome to the Image Message Hider!")
    image_path = input("Enter the path to the image: ")
    secret_message = input("Enter the secret message you want to hide: ")
    output_path = input("Enter the path to save the modified image: ")
    hide_message_in_image(image_path, secret_message, output_path)
    print(f"Message hidden in {output_path}")

if __name__ == "__main__":
    main()
```

Reference

- Aslam, M. A., Rashid, M., Azam, F., Abbas, M., Rasheed, Y., Alotaibi, S. S., & Anwar, M. W. (2022, January). Image steganography using least significant bit (lsb)-a systematic literature review. In *2022 2nd International Conference on Computing and Information Technology (ICCIT)* (pp. 32-38). IEEE.

- Ogundokun, R. O., & Abikoye, O. C. (2021). A safe and secured medical textual information using an improved LSB image steganography. *International Journal of Digital Multimedia Broadcasting*, *2021*(1), 8827055.

- Rustad, S., Syukur, A., & Andono, P. N. (2022). Inverted LSB image steganography using adaptive pattern to improve imperceptibility. *Journal of King Saud University-Computer and Information Sciences*, *34*(6), 3559-3568.

- Subramanian, N., Elharrouss, O., Al-Maadeed, S., & Bouridane, A. (2021). Image steganography: A review of the recent advances. *IEEE access*, *9*, 23409-23423.

- Verma, A. K., & Sarkar, T. (2024, May). Utilizing Imaging Steganographic Improvement using LSB & Image Decoder. In 2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE) (pp. 144-150). IEEE.