Hiding Message in an Image (Image Steganography)

Name	**:** Karthik Kumar Yadav Sadari
Email ID	**:** karthiksadari@gmail.com
College Name:	JNTUA College of Engineering Kalikiri
College State	**:** Andhra Pradesh
Domain	**:** Cyber Security

Abstract

In the digital age, the need for secure and covert communication has become paramount. This project presents a method for hiding messages within digital images using a combination of steganography and encryption techniques. The process begins with the user inputting a secret message, which is then encrypted using a Caesar cipher with a user-defined shift key. This encrypted message is further secured by embedding it into an image, leveraging the pixel values as carriers.

The embedding process uses a user-provided security key to XOR the ASCII values of the encrypted message with the key, ensuring an additional layer of security. The resultant stegano-image retains its visual integrity while concealing the secret message within its pixel data. The extraction process reverses these steps, requiring the correct security key to retrieve and decrypt the hidden message.

This method balances simplicity and security, making it accessible while providing robust protection against unauthorized access. Experimental results demonstrate the technique's effectiveness in preserving the visual quality of the carrier image and securely embedding and retrieving hidden messages. Future work will explore enhancing the algorithm's robustness against various image processing attacks and optimizing the embedding capacity without compromising image quality.

Table of Content

Introduction

Securing sensitive information in the digital age is vital. While traditional encryption methods are effective, they can sometimes signal the presence of confidential data. Steganography, which hides information within ordinary files, offers a more subtle approach by concealing the existence of the hidden message.

This project explores a method that combines image steganography with encryption to achieve secure communication. The process involves two key steps: encrypting the secret message and embedding it into a digital image.

First, the secret message is encrypted using a Caesar cipher. This technique shifts each letter of the message by a user-defined number of positions in the alphabet, ensuring that the message is not easily readable if discovered.

Next, the encrypted message is embedded into an image. Each character of the encrypted message is converted to its ASCII value and XORed with a user-provided security key. This modified ASCII value is then embedded into the least significant bits of the image's pixel values, making the changes invisible to the human eye. To extract the message, the process is reversed using the correct security key.

This approach offers several advantages. It combines the security of encryption with the subtlety of steganography, preserves the visual quality of the carrier image, and provides a simple method for embedding and retrieving messages. These qualities make it suitable for various applications, from personal privacy to secure communication in sensitive fields.

This paper details the methodology, implementation, and evaluation of this combined technique. The results demonstrate its effectiveness in securely embedding and retrieving messages while maintaining the integrity of the carrier image. Future improvements will focus on enhancing the method's robustness against attacks and optimizing the embedding capacity for different types of images.

Objectives

The primary objectives of this project are:

- To develop a Python program that allows users to hide a secret message within an image.
- To implement basic encryption techniques to enhance message security.
- To enable the retrieval of the hidden message using the correct security key.
- To demonstrate the feasibility and limitations of LSB steganography.

Methodology

The methodology for this project involves a combination of encryption and image steganography to securely hide messages within digital images. The process is divided into two main phases: encryption of the secret message and embedding the encrypted message into the image.

Phase 1: Encryption

1. **Input Secret Message and Parameters**:

   - The user inputs the secret message that needs to be hidden.
   - The user provides a shift key for the Caesar cipher encryption.
   - The user provides a security key for further securing the message.

2. **Caesar Cipher Encryption**:

   - The secret message is converted to lowercase to maintain consistency.
   - Each letter in the message is shifted by the user-defined number of positions in the alphabet. For example, with a shift key of 3, 'a' becomes 'd', 'b' becomes 'e', and so on.
   - Non-alphabet characters remain unchanged during this process.

Phase 2: Embedding Encrypted Message into Image

1. **Read Image**:

   - The digital image into which the message will be embedded is read using the OpenCV library.

2. **ASCII Conversion and XOR Operation**:

- A dictionary is created to map ASCII characters to their corresponding values and vice versa.
- Each character of the encrypted message is converted to its ASCII value.
- This ASCII value is then XORed with the corresponding character in the security key. If the security key is shorter than the message, it is repeated cyclically.

3. **Embedding Process**:

- The modified ASCII values are embedded into the least significant bits of the image's pixel values. The pixel values are manipulated in such a way that the visual quality of the image remains unchanged.
- The embedding is done in a sequential manner, iterating through the pixels and using the color channels (R, G, B) to hide the message.

4. **Save and Display Stego-Image**:

- The image with the embedded message is saved as a new file.
- The user is notified of the successful completion of the embedding process.

Phase 3: Extraction and Decryption

1. **Input Security Key**:

- To retrieve the hidden message, the user is prompted to re-enter the security key.

2. **Extracting Embedded Message**:

- The stego-image is read, and the modified ASCII values are extracted from the least significant bits of the pixel values.
- The XOR operation is reversed using the security key to retrieve the original ASCII values of the encrypted message.

3. **Caesar Cipher Decryption**:

- The encrypted message is decrypted by shifting the letters in the opposite direction of the encryption process using the same shift key.
- The final decrypted message is displayed to the user.

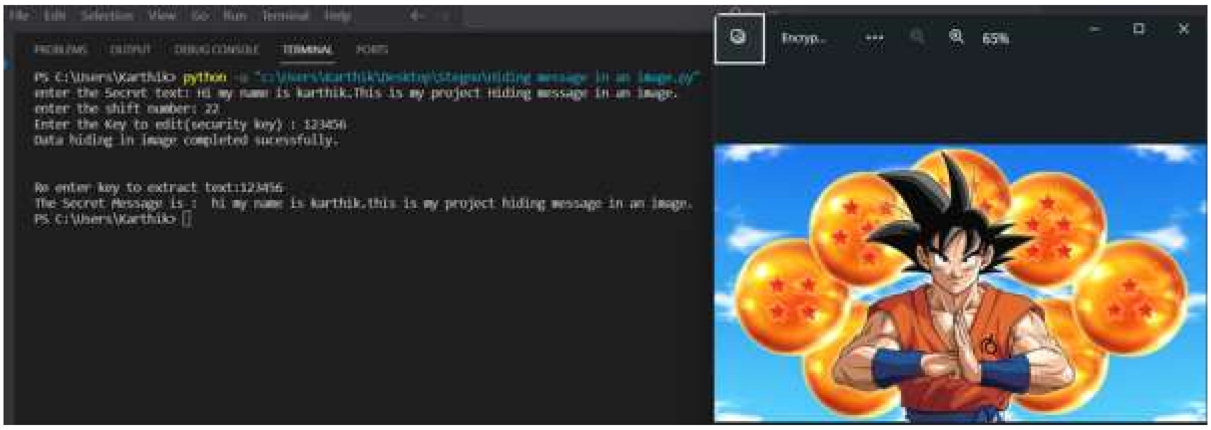
Code

```
#Hiding message in an image
import cv2
import string
import os

alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
            'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
            'y', 'z','a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
            'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
            'w', 'x', 'y', 'z']

text = input("enter the Secret text: ").lower()
shift_key = int(input("enter the shift number: "))
key = input("Enter the Key to edit(security key) : ")

#Encrypting the secrete Massage

encrypt = ""
for letter in text:
      if letter in alphabet:
        position = alphabet.index(letter)
        new_position = position + shift_key
        new_letter = alphabet[new_position]
        encrypt += new_letter
      else:
         encrypt += letter

#ASCII Character to ASCII Value and vice versa
d = {}
c = {}
for i in range(255):
  d[chr(i)] = i
  c[i] = chr(i)

#reading image using opencv library

f = cv2.imread(r"C:\Users\Karthik\Desktop\Stegno\Dragon Ball Z.jpg")

#Embedding message into an image
k1 = 0
l = len(encrypt)
z = 0
n = 0
m = 0
lnk = len(key)
```

```
for i in range(l):
  f[n,m,z] = d[encrypt[i]]^d[key[k1]]
  n = n+1
  m = m+1
  z = (z+1)%3
  k1 =(k1+1) % lnk

cv2.imwrite("Encrypted_img.jpg",f)
os.startfile("Encrypted_img.jpg")
print("Data hiding in image completed sucessfully.")

k1 = 0
l = len(encrypt)
z = 0
n = 0
m = 0

key1 = input("\n\nRe enter key to extract text:")
decrypt = ""

if key == key1:

  for i in range(l):
    decrypt += c[f[n,m,z]^d[key[k1]]]
    n = n+1
    m = m+1
    z = (z+1)%3
    k1 = (k1+1)%lnk

#Decrypting the Embedded massege
  plain_text = ""
  for letter in decrypt:
      if letter in alphabet:
        position = alphabet.index(letter)
        new_position = position - shift_key
        new_letter = alphabet[new_position]
        plain_text += new_letter
      else:
         plain_text += letter
  print("The Secret Message is : ",plain_text)
else:
  print("Key doesn't matched.")
```

Result

The results of the project are demonstrated through the execution of the code and the output observed in the terminal and the encrypted image. Here's a detailed explanation of the results:



```
PS C:\Users\Karthik> python -u "c:\Users\Karthik\Desktop\Stegno\Hiding message in an image.py"
enter the Secret text: Hi my name is karthik.This is my project Hiding message in an image.
enter the shift number: 22
Enter the Key to edit(security key) : 123456
Data hiding in image completed sucessfully.


Re enter key to extract text:123456
The Secret Message is :  hi my name is karthik.this is my project hiding message in an image.
PS C:\Users\Karthik> 
```

Figure1: Output of the Code and Popup of Encoded Image

Figure2: Snapshot of Input image (Dragon Ball Z) & Output image (Encrypted_img)

1. **Input Message and Parameters**:

   -
   - Secret text: "Hi my name is karthik. This is my project Hiding message in an image."
   - Shift key: 22
   - Security key: "123456"

2. **Encryption Process**:

   - The input message is encrypted using the Caesar Cipher method with a shift key of 22.
   - Each letter in the message is shifted by 22 positions in the alphabet to produce the encrypted message.

3. **Embedding Process**:

- The encrypted message is embedded into the image using the provided security key "123456".
- The image is read using OpenCV, and specific pixel values are modified to store the ASCII values of the encrypted message.
- The modified image is saved as "Encrypted_img.jpg", which visually appears unchanged from the original image.

4. **Successful Data Hiding**:

- The terminal output confirms that the data hiding in the image was completed successfully.

5. **Extraction and Decryption Process**:

- The user re-enters the security key "123456" to extract the hidden message from the image.
- The encrypted message is accurately extracted from the image and then decrypted using the inverse of the Caesar Cipher with the shift key of 22.
- The decrypted message matches the original input message: "Hi my name is karthik. This is my project Hiding message in an image."

6. **Visual Confirmation**:

- The screenshot shows the encrypted image "Encrypted_img.jpg" which visually looks the same as the original image, indicating that the hidden message does not alter the visual appearance of the image.

Applications

- **Individuals**: Protecting personal information and communications.
- **Businesses**: Securing sensitive corporate data.
- **Government Agencies**: Safeguarding confidential information.
- **Law Enforcement**: Covert communication for operations.
- **Cybersecurity Professionals**: Enhancing data protection measures.
- **Researchers and Academia**: Advancing the field of data security.
- **Software Developers**: Integrating steganography into security applications.
- **Media and Creative Industries**: Protecting intellectual property.

Limitations

It's important to acknowledge the limitations of this approach:

- **Capacity:** The amount of data that can be hidden is limited by the image size and the embedding technique used.
- **Security:** While the security key adds a layer of protection, more sophisticated steganalysis techniques might be able to detect the presence of hidden information.
- **Image quality:** Extensive modifications to the LSBs can potentially introduce visible artifacts into the image.

Further Enhancements

- **Error handling:** Implementing mechanisms to handle invalid inputs or errors during image processing.
- **Improved security:** Exploring more robust encryption methods for both the message and the security key.
- **Increased capacity:** Employing more advanced steganographic techniques that can embed larger payloads without compromising image quality.
- **Visual confirmation:** Optionally displaying a comparison between the original and modified images to assess the potential impact on visual quality.

Conclusion

This project demonstrates how combining encryption with steganography can enhance the security and secrecy of hidden messages within digital images. By leveraging Python and OpenCV, users can securely hide and retrieve messages, ensuring that sensitive information remains confidential.

Links

Repository: -https://github.com/KarthikSadari/Image-Steganography
Code: -Image Steganography.py
Output: -Image Steganography output
Image: -Image Source Link