CS4475 | Project 2

Steganography

Thomas Dean, Noah Flanagan, Annarose Hilhorst, Caleb Jones

Table of Contents

High Level

Our project aims to tackle the concept of steganography, the process of hiding data within an image. The Python script performs basic Least Significant Bit (LSB) steganography to hide and extract messages in an image. It includes methods for encoding a message into an image and decoding a message from an image. Also, it uses XOR with a hidden key to make it more secretive. Have fun encoding and decoding!

Additionally, the code provides a command-line interface for users to interact with the tool and perform the following tasks:

1. Hide a file within an image
2. Fetch (extract) a hidden file from an image
3. Hide and then fetch a file from an image for verification

For fun, we added a script for both Linux/MacOS (bash.png) and Windows (windows.jpg) that appears as an image file in name, but will call encode.py to run our code.

Low Level

1. `display(image)`

- Purpose: Display an image using OpenCV.
- How:
    - `cv2.namedWindow("image", cv2.WINDOW_NORMAL)`: Creates a resizable window.
    - `cv2.imshow("image", image)`: Displays the image.
    - `cv2.waitKey(0)`: Waits indefinitely for a key event.
    - `cv2.destroyAllWindows()`: Closes all OpenCV windows.

2. `read_bytes(b_msg)`

- Purpose: Converts bytes to a readable string.
- How: Iterates through the byte message, converting each byte to a character and concatenating to form a string.

```
def read_bytes(b_msg):
    rtn = ''
    for byte in b_msg:
        rtn += chr(byte)
```

3. `encrypt_int(integer)`

- Purpose: Encrypts an integer using XOR with a private key.
- How: XORs the integer with a predefined key (15723480).

```
def encrypt_int(integer):
    private_key = 15723480
    return integer ^ private_key
```

4. `hide(image, b_message)`

- Purpose: Hides a byte message within an image.
- How:
    - Calculates the total storage space in the image.

```
dim = len(image.shape)
```

    - Checks if the image has enough space to store the message.

```
store_space = image.shape[0]
for i in range(1, dim):
    store_space *= image.shape[i]

if store_space < len(b_message) * 8:
    raise ValueError("Image must have enough pixels to store message")
```

    - Converts each byte of the message to bits.

```
for byte in b_message:
    # https://blog.finxter.com/5-best-ways-to-convert-python-bytes-to-bits/
    temp = list(map(int, bin(byte)[2:].zfill(8)))
    bits += temp
```

- Calls `bit_insert` to insert the bits into the image.

```
return bit_insert(image, bits, dim)
```

5. `bit_insert(image, bits, dim)`

- Purpose: Inserts bits into the least significant bits (LSBs) of the image.
- How:
  - Creates a mask to zero out the LSB.

```
mask = 0b11111110
```

  - Iterates through the image, modifying the LSB of each pixel to store the bits.

```
if dim == 3:
    # go by x, y, then color
    for color in range(image.shape[2]):
        for x in range(image.shape[1]):
            for y in range(image.shape[0]):
                pixel = image[y, x, color] & mask
                rtn[y, x, color] = pixel + bits[b_i]
                b_i += 1
                if b_i >= len(bits):
                    return rtn
else:
    for x in range(image.shape[1]):
        for y in range(image.shape[0]):
            pixel = image[y, x] & mask
            rtn[y, x] = pixel + bits[b_i]
            b_i += 1
            if b_i >= len(bits):
                return rtn
```

  - Handles both color and grayscale images.

6. `decrypt(image, msg_len)`

- Purpose: Extracts a hidden message from an image.
- How:
  - Calls `bit_extract` to retrieve bits from the image.
  - Converts the bits back into bytes and then into a byte message.

```
for i in range(0, len(bits), 8):
    res = int(''.join(map(str, bits[i:i+8])), 2)
    b_list.append(res)

return bytes(b_list)
```

7. `bit_extract(image, msg_len)`

- Purpose: Extracts bits from the LSBs of the image.
- How:
    - Creates a mask to isolate the LSB.
    - Iterates through the image, extracting the LSB of each pixel.

```
if dim == 3:
    # go by x, y, then color
    for color in range(image.shape[2]):
        for x in range(image.shape[1]):
            for y in range(image.shape[0]):
                bits.append(image[y, x, color] & mask)
                b_count += 1
                if b_count >= msg_len*8:
                    return bits
# grayscale
else:
    for x in range(image.shape[1]):
        for y in range(image.shape[0]):
            bits.append(image[y, x])
            b_count += 1
            if b_count >= msg_len*8:
                return bits
```

    - Handles both color and grayscale images.

8. `user_file_hide(image)`

- Purpose: Hides a file within an image.
- How:
    - Prompts the user for the file path.

```
while True:
    test_msg = input('Please enter the file path you would like to store in the image.\n')
```

- Reads the file as bytes.

```
filetype = test_msg.split(".")[-1]
f = open(test_msg, "rb")
file = f.read()
```

- Calls `hide` to embed the file within the image.
- Displays an error if the image is too small.

9. `user_file_decrypt(secret, key, filetype)`

- Purpose: Decrypts a file from a hidden image.
- How:
  - Extracts the byte message using `decrypt`.

```
b_msg = decrypt(secret, key)
```

  - Saves the extracted message as a file with the specified file type.

```
# https://www.geeksforgeeks.org/create
file_path = "./{}".format(filetype)
```

10. `user_both(image)`

- Purpose: Hides and then immediately extracts a file from an image.
- How:
  - Prompts the user for the file path.

```
while True:
    test_msg = input('Please enter file path you would like to store in the image.\n')
```

  - Hides the file within the image using `hide`.

```
secret = hide(image, file)
```

- Extracts the file from the image using `decrypt`.

```
b_msg = bytes(decrypt(secret, len(file)))
```

- Saves the extracted file using the write function in python.

11. main()

- Purpose: Provides a user interface for the steganography tool.
- How:
  - Prompts the user for an operation (hide, fetch, both, or quit).

```
while True:
    count = 0
    opt = input('''Welcome! Please select what you want to do:
    (1) Hide file in image
    (2) Fetch file from image
    (3) Hide and fetch file
    (4) Quit
    ''')
```

  - Depending on the choice, it calls the appropriate function.

```
match opt:
    case '1':
        i_path = input("Please specify the source image's path: ")
        image = cv2.imread(i_path)

        secret = user_file_hide(image)
        print('File hidden in image.')
        name = 'secret{}.png'.format(count)
        print(name, 'saved in current directory!')
        count += 1
        cv2.imwrite(name, secret)
    case '2':
        secret_path = input('Please enter the file path to the image to fetch from: ')
        secret = cv2.imread(secret_path)

        key = int(input('Please enter the key: '))

        filetype = input('Please enter the file name to store result: ')

        user_file_decrypt(secret, encrypt_int(key), filetype)
        print('File saved to current directory!')
    case '3':
        i_path = input("Please specify the source image's path: ")
        image = cv2.imread(i_path)
        secret = user_both(image)
        y = input('Would you like to compare ? (y/n)\n')

        if y == 'y':
            display(image)
            display(secret)
    case '4':
        return
    case _:
        continue
```

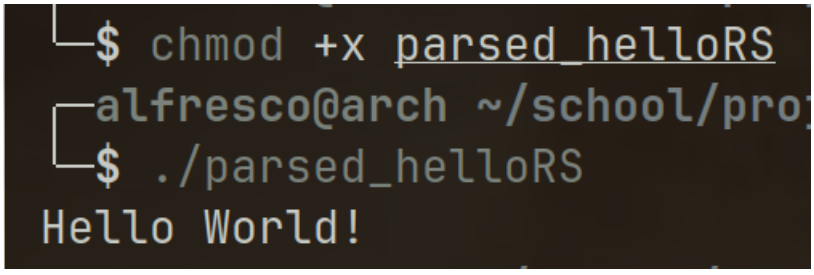
- Handles image reading and writing using OpenCV functions.

Results

Original Image:

Encoded image:

Using the decrypt function:



Grading Breakdown

15 - Creativity | 45 - Technical

Attributions

Generative AI was not consulted in the making of this project.

Converting python bytes to bits:
https://blog.finxter.com/5-best-ways-to-convert-python-bytes-to-bits/

Creating a new text file in python:
https://www.geeksforgeeks.org/create-a-new-text-file-in-python/

XOR Cipher:
https://www.geeksforgeeks.org/xor-cipher/