Secure Data Communication using Steganography

Team Members:

- Grace Selina Pagadala
- Jasmika Vemulapalli
- Poojasri Gadde
- Sri Haneesha Davuluri

- ✓ Objective
- ✓ Motivation
- ✓ What is Steganography
- ✓ Steganography Vs Cryptography
- ✓ Types of Steganography
- ✓ Image Steganography
- ✓ Techniques
- ✓ Proposed Methodology
- ✓ Code Snippets
- ✓ Results
- ✓ Comparision
- ✓ References

Contributions

Grace Selina Pagadala: Encryption Code, **Testing**

Jasmika Vemulapalli: Novel Idea, Decryption Code

Poojasri Gadde: Algorithm, **Decryption** Code

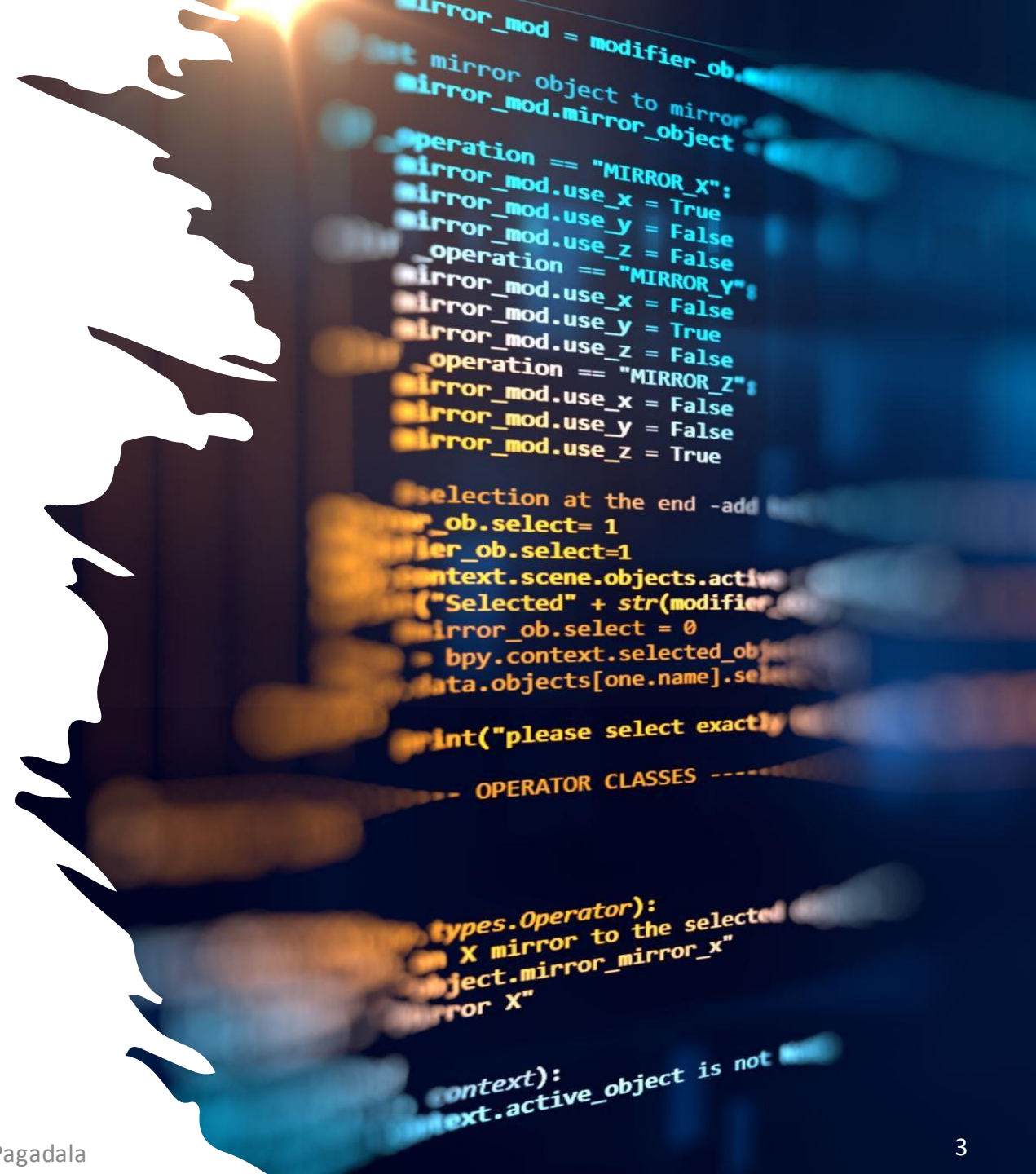
Sri Haneesha Davuluri: **Encryption** Code, Overall Code Integration

Contribution Ratio of Team Members: *25% each*

Objective

The primary objective of this project is to implement a secure data communication system using steganography.

The project aims to hide secret data within images using Python and the PIL (Pillow) library.



Motivation

Data breaches and cyber attacks are becoming more frequent.

Over time, there has been a significant increase in the requirement for security and privacy.

Lower threat risk results from improved security.

Data hiding from attackers or malicious users can result in stronger security measures.

Poojasri Gadde Jasmika Vemulapalli Sri Haneesha Davuluri Grace Selina Pagadala

Steganography

- Steganography involves concealing sensitive information within another medium to prevent detection.
- Steganography hides the message's existence, unlike encryption, which aims to render the message content unreadable to unauthorized users.
- Computer Steganography is based on 2 principles:
  1. The files containing digitized images or sounds can be altered to a certain extent without losing their functionality.
  2. The second principle involves exploiting human difficulty in discerning slight changes in image color or sound quality.

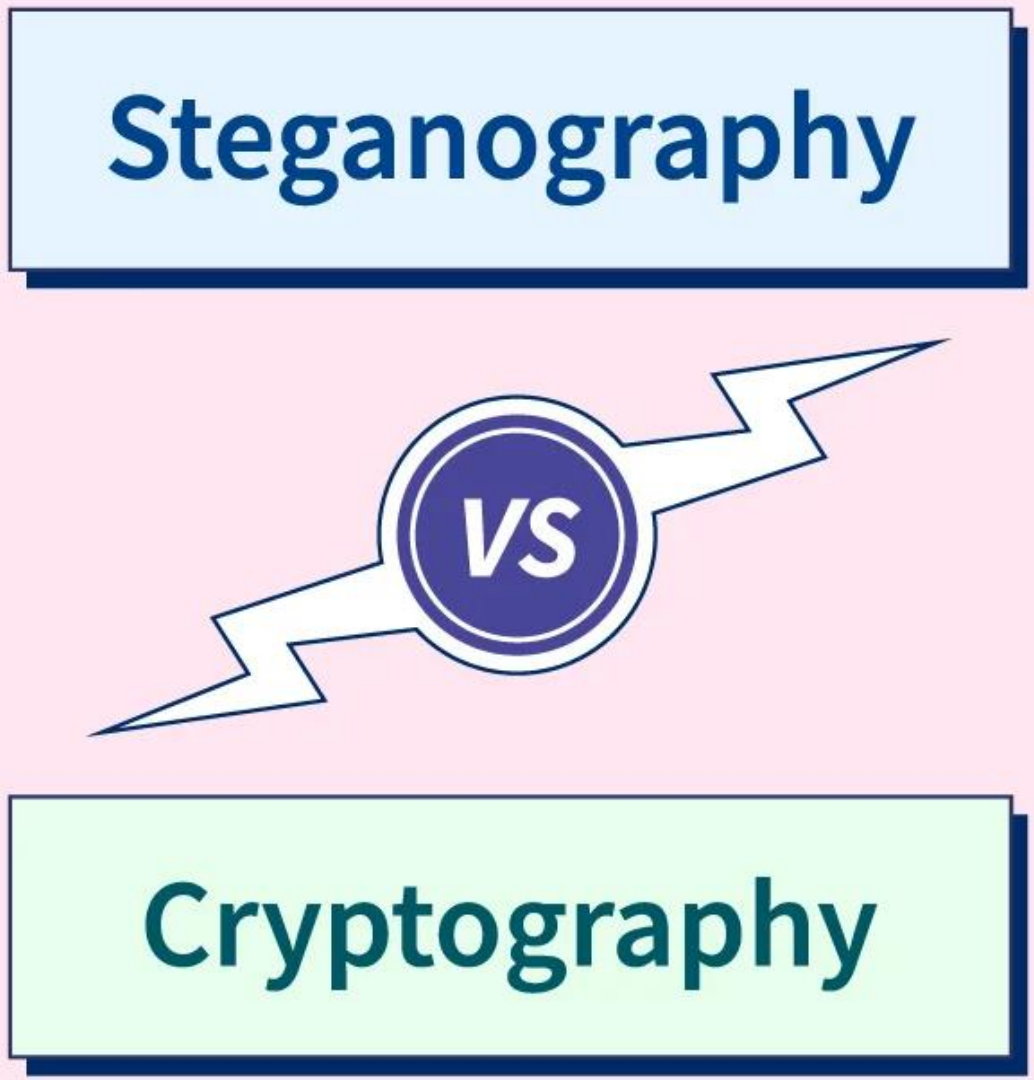
Steganography and Cryptography are both techniques used for information security, but they differ in their fundamental approaches and objectives.

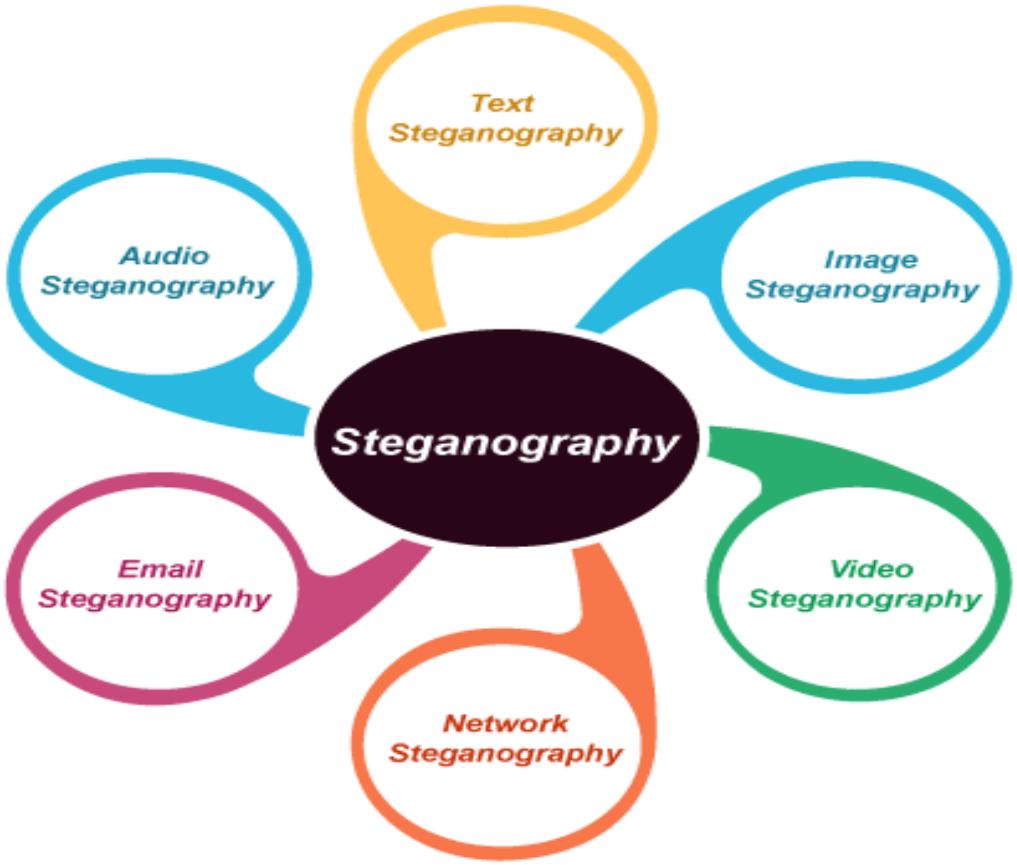
1. Concealment Vs. Encryption:

- Steganography:
    1. Hides message existence.
    2. Conceals within media covertly.
    3. Focuses on covert communication.
- Cryptography:
    1. Secures message content.
    2. Encrypts information for confidentiality.
    3. Emphasizes content protection.

2. Detection Vs. Decryption:

- Steganography:
    1. Detection finds concealed information.
    2. Aims to identify covert communication.
- Cryptography:
    1. Decryption transforms ciphertext to plaintext.
    2. Prevents unauthorized access to content.





TYPES OF STEGANOGRAPHY

Image Steganography

It is the technique of hiding the data within the image in such a way that prevents the unintended user from the detection of the hidden messages or data.

Applications:

- Secure Private Files and Documents.
- Hide Passwords and Encryption Keys.
- Transport Highly Private Documents between International Governments.
- Transmit message/data without revealing the existence of available message.

Techniques

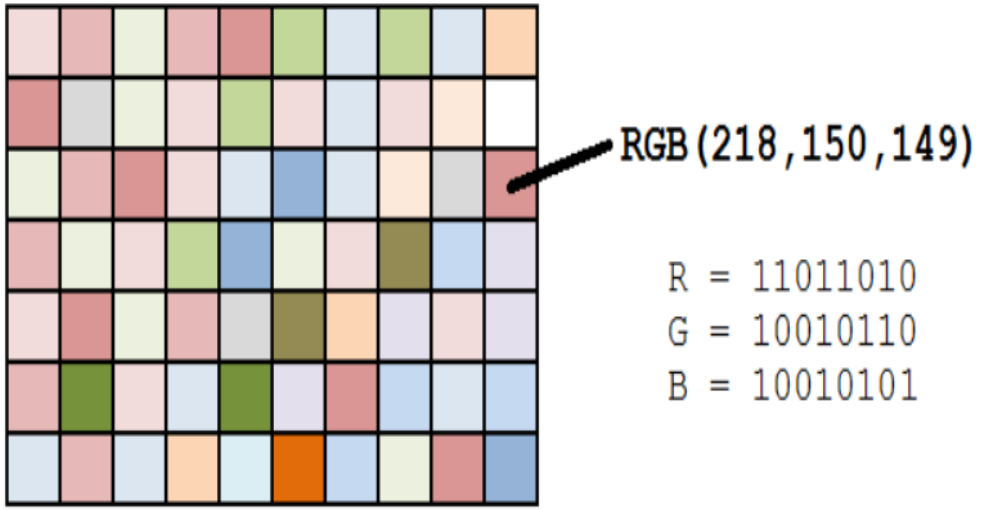
Least Significant Bit(LSB) Substitution

- Images are made up of lots of little dots called pixels. Each pixel is represented as 3 bytes - one for Red, one for Green and one for Blue.

  11011010 10010110 10010101

  218 150 149

- Each byte is interpreted as an integer number, which is how much of that color is used to make the final color of the pixel.
- The difference between two colors that differ by one bit in either one red, green or blue value is impossible to detect for a human eye.
- So, we can change the least significant(last) bit in a byte, we either add or subtract one or more values from the value it represents.
- This means we can overwrite the last bit in a byte without affecting the colors it appears to be.
- In this method, we can take the binary representation of the hidden data and overwrite the LSB of each byte within the cover image.



Advantages

LSB substitution is straightforward and easy to implement.

The computational complexity of LSB substitution is relatively low.

Preserves the Image Quality.

LSB substitution can provide a high payload capacity, allowing for the hiding of a significant amount of data within an image.

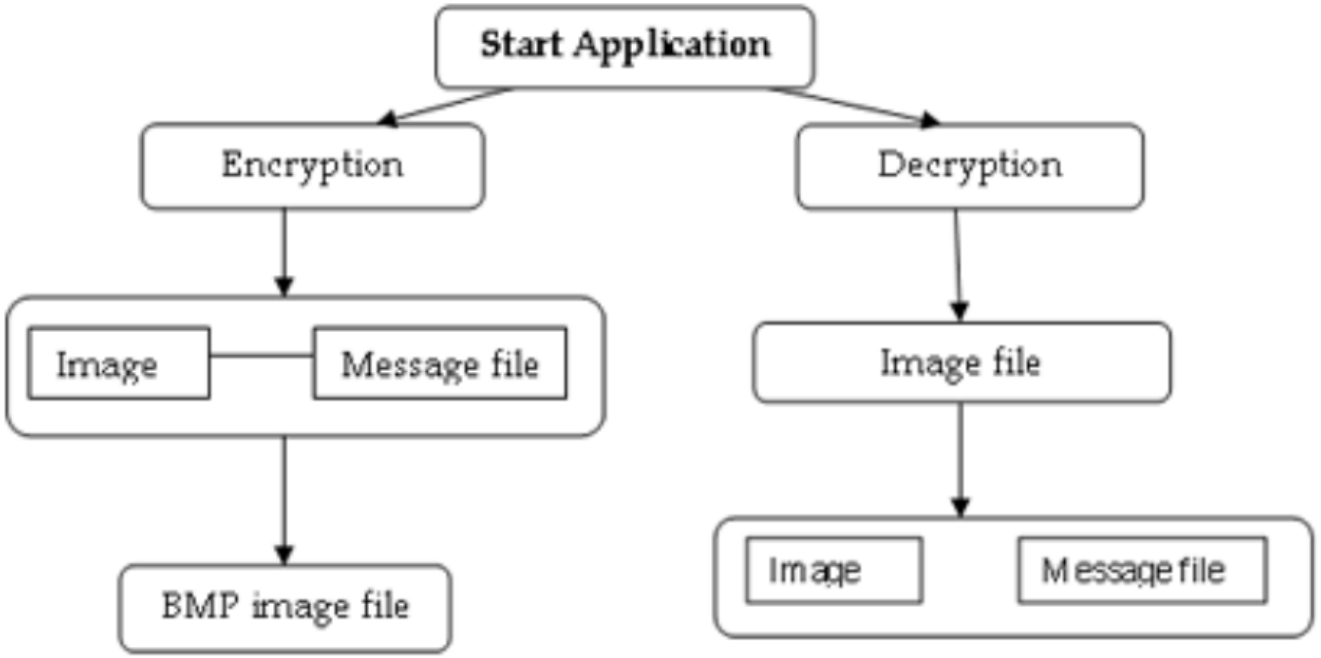
LSB substitution is versatile and can be applied to various types of digital media, including images, audio, and text.

LSB substitution can be applied to various image formats like BMP, PNG, and JPEG without significant modifications.

Some steganographic techniques may suffer from issues related to image compression. LSB substitution, when applied conservatively, is less susceptible to compression artifacts.

Proposed Methodology

The graphical representation of the Steganography system:



Code Snippets

```
main()
```

```
:: Welcome to Steganography using Steganography:
1. Encode the data
2. Decode the image
1
Enter image file name: demo.png
Enter data to be encoded in the image: Data security means protecting digital data, such
Time taken for encryption: 0.00302 seconds
Enter the name of new image file to the decoded image: dec.png
```

Encoding Algorithm

The ASCII value of each character in the data is extracted and transformed to an 8-bit binary.

There are a total of 3*3=9 RGB values in the three pixels that are read at once. One character is stored using the first eight RGB values after being transformed into an 8-bit binary.

The binary data and related RGB value are contrasted. The RGB value is translated into odd if the binary digit is 1, and into even otherwise.

The ninth value decides whether or not to scan more pixels. The ninth pixel becomes even if there is more information to be read, encoded, or decoded. Otherwise, make it unusual if you want to stop reading pixels further.

Continue doing this until the image has encoded all of the data.

Poojasri Gadde Jasmika Vemulapalli Sri Haneesha Davuluri Grace Selina Pagadala

Encoding Code Snippet

```
def encode():
    # Get the filename of the image to be encoded
    image_filename = input("Enter image file name: ")

    # Open the image
    image = Image.open(image_filename, 'r')

    # Display the image
    image.show()

    # Get the data to be encoded
    data = input("Enter data to be encoded in the image: ")

    # Raise an error if the data is empty
    if len(data) == 0:
        raise ValueError('Data is empty')

    # Create a copy of the image to modify
    new_image = image.copy()

    # Record the start time for encryption
    start_time = time.time()

    # Encode the data in the new image
    encode_enc(new_image, data)
    # Record the end time for encryption
    end_time = time.time()
    # Print the time taken for encryption
    print(f"Time taken for encryption: {end_time - start_time:.5f} seconds")

    # Get the filename of the new image
    new_image_filename = input("Enter the name of new image file to the decoded image: ")

    # Save the new image with the appropriate file format
    new_image.save(new_image_filename, str(new_image_filename.split(".")[1].upper()))
```

```python
def encode_enc(image, data):
    # Get the width of the image
    image_width = image.size[0]

    # Start at the top-left corner of the image
    x_coord = 0
    y_coord = 0

    # Loop over each modified pixel and add it to the new image
    for modified_pixel in modPix(image.getdata(), data):

        # Put the modified pixel in the new image
        image.putpixel((x_coord, y_coord), modified_pixel)

        # Move to the next row if at the end of the current row
        if x_coord == image_width - 1:
            x_coord = 0
            y_coord += 1
        else:
            x_coord += 1
```

```python
def modPix(pix, data):
    # Convert data to a list of binary strings
    binary_data = genData(data)
    data_len = len(binary_data)    # Get the length of the data list
    pixel_iter = iter(pix) # Get an iterator for the image pixel data
    # Loop through each set of 3 pixels for each binary string in the data list
    for i in range(data_len):

        # Extract the next 3 pixels from the image pixel data
        pixel_values = [value for value in pixel_iter.__next__()[:3] + pixel_iter.__next__()[:3] + pixel_iter.__next__()[:3]]
        # Modify each pixel value based on the binary string
        for j in range(0, 8):
            if (binary_data[i][j] == '0' and pixel_values[j] % 2 != 0):
                pixel_values[j] -= 1
            elif (binary_data[i][j] == '1' and pixel_values[j] % 2 == 0):
                if (pixel_values[j] != 0):
                    pixel_values[j] -= 1
                else:
                    pixel_values[j] += 1

        # Modify the eighth pixel of each set to indicate whether to continue reading or stop
        if (i == data_len - 1):
            # If this is the last set of pixels, set the eighth pixel to 0 or 1 based on the parity of the current value
            if (pixel_values[-1] % 2 == 0):
                if (pixel_values[-1] != 0):
                    pixel_values[-1] -= 1
                else:
                    pixel_values[-1] += 1
        else:
            # If this is not the last set of pixels, set the eighth pixel to 0 if it is odd
            if (pixel_values[-1] % 2 != 0):
                pixel_values[-1] -= 1

        # Yield the modified pixel values in sets of 3
        pixel_values = tuple(pixel_values)
        yield pixel_values[0:3]
        yield pixel_values[3:6]
        yield pixel_values[6:9]
```

Encryption Images

Plain Image

Encoded Image

Decoding

1. Three pixels are read at a time once more. The first eight RGB values inform us of the secret data, and the ninth value indicates whether we should proceed.
2. For the first eight numbers, the binary bit is 1 for odd values and 0 otherwise for the first eight values.
3. The bits are joined together to form a string, and every three pixels, or one character, corresponds to a byte of secret data.
4. We continue reading pixels if the ninth number is even.

Decoding Code Snippet

```
# Decode the data in the image
def decode():
     # Get the filename of the image to be decoded
    image_filename = input("Enter image name file to be decoded: ")
    # Record the start time for decryption
    start_time = time.time()
    # Open the image
    image = Image.open(image_filename, 'r')
    # Initialize an empty string to store the decoded data
    decoded_data = ''
    # Get an iterator for the image data
    image_data = iter(image.getdata())

    while True:
        # Get the RGB values of the next three pixels
        pixels = [value for value in image_data.__next__()[:3] +image_data.__next__()[:3] +
                                image_data.__next__()[:3]]

        # Convert the least significant bit of each pixel to binary and append to binstr
        binstr = ''
        for i in pixels[:8]:
            if i % 2 == 0:
                binstr += '0'
            else:
                binstr += '1'
        # Convert the binary string to a character and append to the decoded data
        decoded_data += chr(int(binstr, 2))
        # Check if the last pixel's least significant bit is 1
        if pixels[-1] % 2 != 0:
            # Record the end time for decryption
            end_time = time.time()
            # Print the time taken for decryption
            print(f"Time taken for decryption of the image: {end_time - start_time:.5f} seconds")
            return decoded_data
```

Decryption Code Snippet

Output from Encrypted Image:

```
main()
```

```
:: Welcome to Steganography using Steganography:
1. Encode the data
2. Decode the image
2
Enter image name file to be decoded: dec.png
Time taken for decryption of the image: 0.03548 seconds
Decoded Word from the image:  Data security means protecting digital data, such as those in a database,
```

Time Complexity for Encryption

Pixels/words	<10	10-50	250	500	1000
256	0	0.0027	0.0159	0.0259	0.0754
720	0	0.0012	0.0165	0.0218	0.0349
1080	0.0011	0.0027	0.0097	0.0209	0.0340
1440	0.0011	0.0035	0.0110	0.0286	0.0606
2048	0.0015	0.0048	0.0137	0.0368	0.0914

Table 1: Time taken in seconds to encrypt the message in steganography.

Time Complexity for Decryption

Pixels/words	<10 words	10-50	250	500	1000
256	0.0096	0.0145	0.0116	0.0164	0.0194
720	0.0128	0.0126	0.0146	0.0172	0.0225
1080	0.0456	0.0468	0.0497	0.0723	0.0863
1440	0.0373	0.0646	0.0685	0.0692	0.0912
2048	0.0490	0.0753	0.0799	0.0815	0.1443

Table 2: Time taken in seconds to decrypt the message in steganography

What if the Message size is bigger?

- The time complexity of the encoding process can be considerably impacted by the size of the message. The message data is compressed during the encoding process so that it can be transmitted more quickly and with less storage space. However, when the message size grows, the compression algorithm will need more computing power to compress the data, which will make the process more time-consuming.
- Moreover, problems with memory management can arise from greater message sizes. If the message size is too large, it might not fit within the memory that needs to be allocated by the encoding process to store the compressed message data. This may cause slower performance or, in some circumstances, crashes.

Conclusion

To human eyes, the new image (dec.png) is exactly like the original image (demo.png). Human eyes cannot detect the little shift in pixel values. A human cannot tell the difference between the two photos. If a computer doesn't have access to the original image to compare it to, it will be exceedingly difficult for it to recognize the image as a Steganography image.

Future Works

- Explore advanced steganographic techniques for increased security.

For example, Triple-A technique uses the same principle of LSB, where the secret is hidden in the least significant bits of the pixels, with more randomization in selection of the number of bits used and the color channels that are used. This randomization is expected to increase the security of the system. It measures the intensity of the pixel and then hides data by random pixel selection with a goal to hide maximum data in each pixel.

- Develop a graphical user interface (GUI) for user-friendly interactions.

References

1. N. Subramanian, O. Elharrouss, S. Al-Maadeed and A. Bouridane, "Image Steganography: A Review of the Recent Advances," in *IEEE Access*, vol. 9, pp. 23409-23423, 2021, doi: 10.1109/ACCESS.2021.3053998.

2. W. Liu and J. Wang, "Research on image steganography information detection based on support vector machine," *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, Xi'an, China, 2021, pp. 631-635, doi: 10.1109/ICSP51882.2021.9408671.

3. A. A. J. Altaay, S. B. Sahib and M. Zamani, "An Introduction to Image Steganography Techniques," *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, Kuala Lumpur, Malaysia, 2012, pp. 122-126, doi: 10.1109/ACSAT.2012.25.

4. J. Qin, Y. Luo, X. Xiang, Y. Tan and H. Huang, "Coverless Image Steganography: A Survey," in *IEEE Access*, vol. 7, pp. 171372-171394, 2019, doi: 10.1109/ACCESS.2019.2955452.

THANK YOU