IMAGE STEGANOGRAPHY SYSTEM

IT-360 Professor

Dr.Manel Abdelkader

Authors

Rayen Rouaissi

Imen Kenza Chouaieb

Mohamed Aziz Belhadj

1.THEORITICAL ASPECTS:

01.STEGANOGRAPHY:

Steganography is the practice of concealing secret information within non-secret information in such a way that it is difficult for outsiders to detect the presence of the hidden information.

02.IMAGE STEGANOGRAPHY:

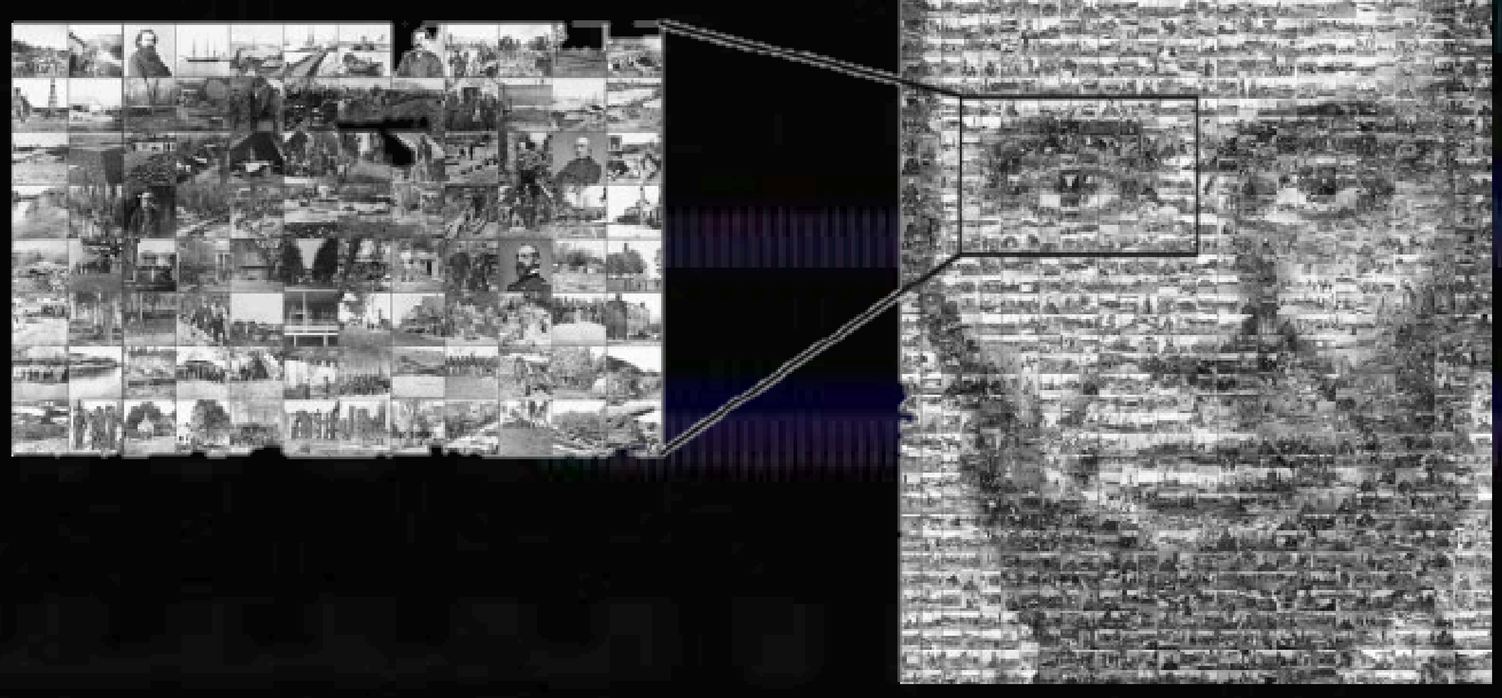
Image steganography is the covert embedding of digital information within an image carrier, where the hidden data is imperceptible to the human visual system and the presence of the information remains statistically undetectable.

- ## MISSION

Our online platform provides a user-friendly solution for secure steganographic processes. Users can easily upload images and embed confidential data using their preferred steganography method. The system ensures the secure concealment of this data within the final image, which users can confidently export, knowing their private information is safely hidden.

- ## VISION:

Steganography offers an extra layer of protection for sensitive data by hiding it within images or other common files. This makes it harder for attackers to detect the information, even if they intercept encrypted data, improving overall data security and privacy.



03.OTHERTYPES:

- **Audio Steganography:** Concealing data within audio files without altering the quality of the sound, often by imperceptibly modifying the frequencies or amplitudes.
- **Video Steganography:** Hiding information within video files by manipulating frames, motion vectors, or other video elements, maintaining the visual integrity of the video.
- **Text Steganography:** Embedding secret messages within text documents by utilizing techniques such as invisible ink, altering spacing, or using nonprintable characters.
- **File Steganography:** Concealing data within various file formats, such as documents, spreadsheets, or executable files, often by exploiting unused space or appending data at the end of the file.
- **Network Steganography:** Concealing data within network protocols or communication channels, allowing covert communication without raising suspicion.

II.MAIN COMPONENTS:

01.STEGANOGRAPHY ALGORITHM:

A steganography algorithm is a set of rules and procedures used to hide secret information within a cover medium, such as an image, audio file, or video. It defines how the secret data is embedded into the cover medium and how it can be extracted later.

02.STEGANOGRAPHY KEY:

The steganography key is a piece of information, such as a passphrase or cryptographic key, that is used to control the embedding and extraction process in steganography. It ensures that only authorized parties with the correct key can access the hidden secret message within the cover medium.

03.COVER IMAGE:

A cover image is the visible or audible medium used in steganography to conceal the secret message. It can be a digital image, audio clip, video file, or any other carrier medium that appears normal and does not raise suspicion regarding the presence of hidden data.

04.SECRET MESSAGE:

The secret message is the confidential information that is embedded within the cover image or other media using steganography techniques. It can be text, files, or any data that needs to be kept confidential and is hidden within the cover medium to prevent unauthorized access or detection.

III.FUNCTIONAL FLOWS:

01.INPUT COVER IMAGE:

The process begins by inputting a cover image, which is the carrier medium used for steganographic embedding. This image should appear normal and not raise suspicion about the presence of hidden data.

02.SELECT STEGANOGRAPHY IMAGE:

Choose a steganography algorithm based on your security requirements and the desired level of imperceptibility. Common algorithms include LSB (Least Significant Bit) embedding, permutation steganography, or more advanced techniques like batch steganography or chaos-based methods.

03.INPUT SECRET MESSAGE:

Provide the secret message or data that you want to hide within the cover image. This can be text, files, or any digital content that needs to be kept confidential.

04.EMBED SECRET MESSAGE:

Use the selected steganography algorithm to embed the secret message into the cover image. The algorithm modifies specific elements of the image, such as pixel values or color channels, to encode the hidden data without significantly altering the visual appearance of the image.

05.GENERATE STEGO IMAGE:

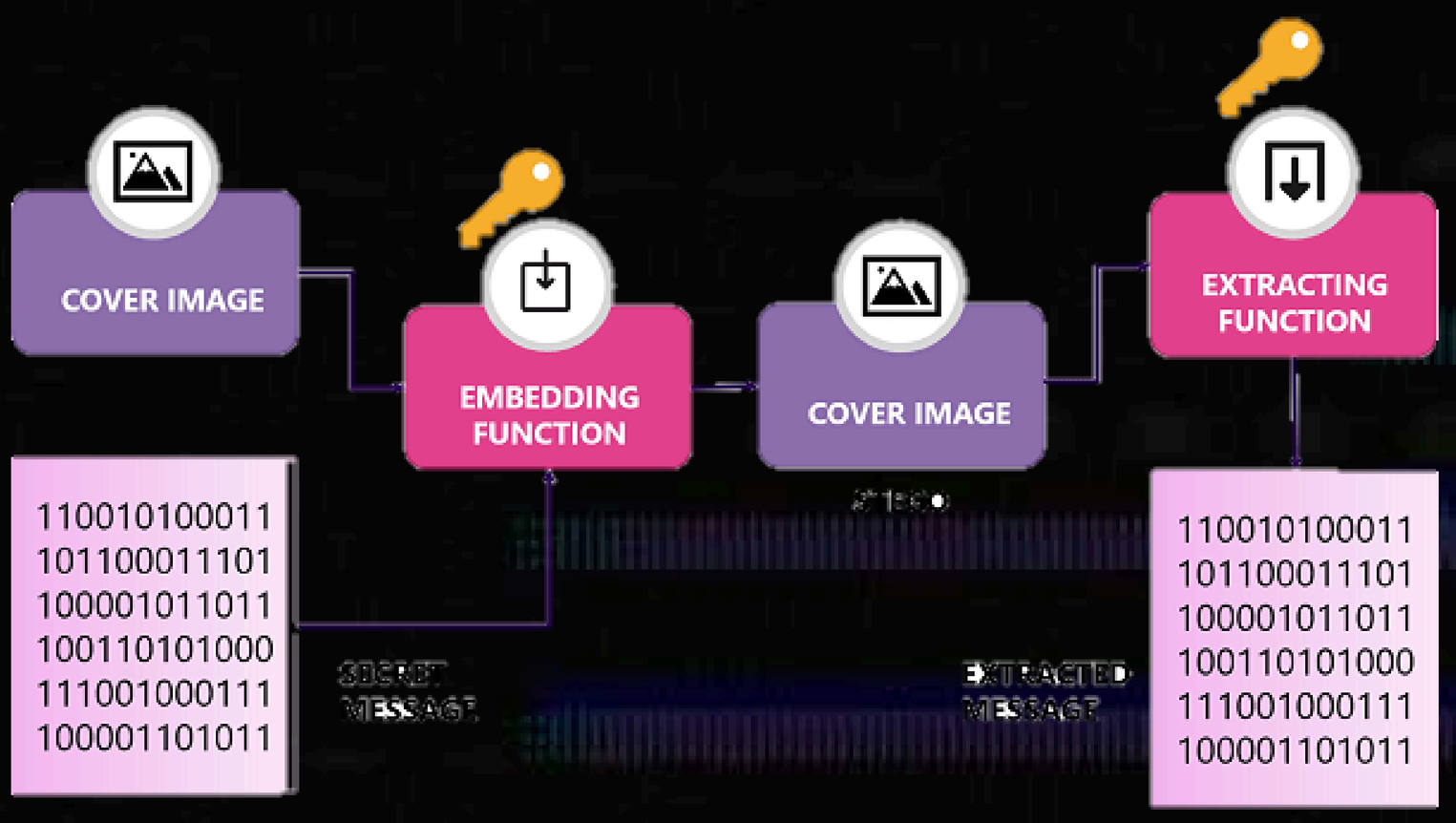
After embedding the secret message, generate the stego image, which is the modified version of the cover image containing the hidden data. The stego image should look similar to the original cover image to maintain imperceptibility

06.EXPORT STEGO IMAGE:

Save or export the stego image, which now includes the concealed secret message. This image can be shared or transmitted like any regular image, without revealing the hidden data to unauthorized individuals.

07.OUTPUT SECRET MESSAGE:

Once the secret message is extracted, it can be decrypted or processed further to access the original confidential information. This completes the functional flow of image steganography, demonstrating the secure embedding and extraction of hidden data within digital images.



IV.REQUIREMENTS:

• IMPERCEPTIBILITY:

The alterations made to the cover image should be imperceptible to the human eye. The steganographic process should not introduce noticeable distortions or artifacts that could raise suspicion.

• CAPACITY:

The steganographic method should have sufficient capacity to embed the desired amount of data within the image without significantly degrading its quality. Higher capacity allows for the concealment of larger messages or data payloads.

• ROBUSTNESS:

The hidden message should remain intact and recoverable even in the presence of common image processing operations, compression algorithms, or other attacks aimed at detecting or removing the embedded data.

• SECURITY:

The steganographic technique should provide security against unauthorized access or detection of the hidden message. This includes resistance to statistical analysis, cryptographic attacks, and other methods used to uncover concealed information.

V.ALGORITHMS:

01.BATCH STEGANOGRAPHY:

- **Main Characteristics:**

Batch steganography is a technique for securely hiding information across multiple covers. It increases secure capacity based on the square root of the number of objects, suggesting not all covers should be used for embedding. It uses pooled steganalysis for reliable detection of steganography in large sets. In adaptive batch steganography, a new strategy determines the sub-batch of cover images for the message. A secure factor is used to evaluate embedding security and calculate payloads for each image. These characteristics can vary with the specific method used.

- **Early stage:**

Steganography has a long history dating back to ancient Greece and Rome. From invisible ink revealed by heat to messages hidden within paintings, steganography has evolved to hide data in digital media. Today it's used for a variety of purposes, from protecting data integrity to hiding secret messages.

- **Advantages:**

- Efficiency: allows for the processing of multiple images in a single operation, saving time and resources.

-Scalability: suitable for situations where large datasets need to be securely transmitted or stored.

- **Limits:**

- Consistency: The effectiveness and security may vary across different images due to varying characteristics of the batch files.

-Detection Risk: Bulk application might increase the risk of detection if the same technique is applied in a detectable pattern.

02. PERMUTATION STEGANOGRAPHY:

- **Main Characteristics:**

This technique involves rearranging the pixels or bits of an image according to a secret key before embedding the secret message. Only someone who knows the key can reverse the permutation and retrieve the message.

- **How it works:**

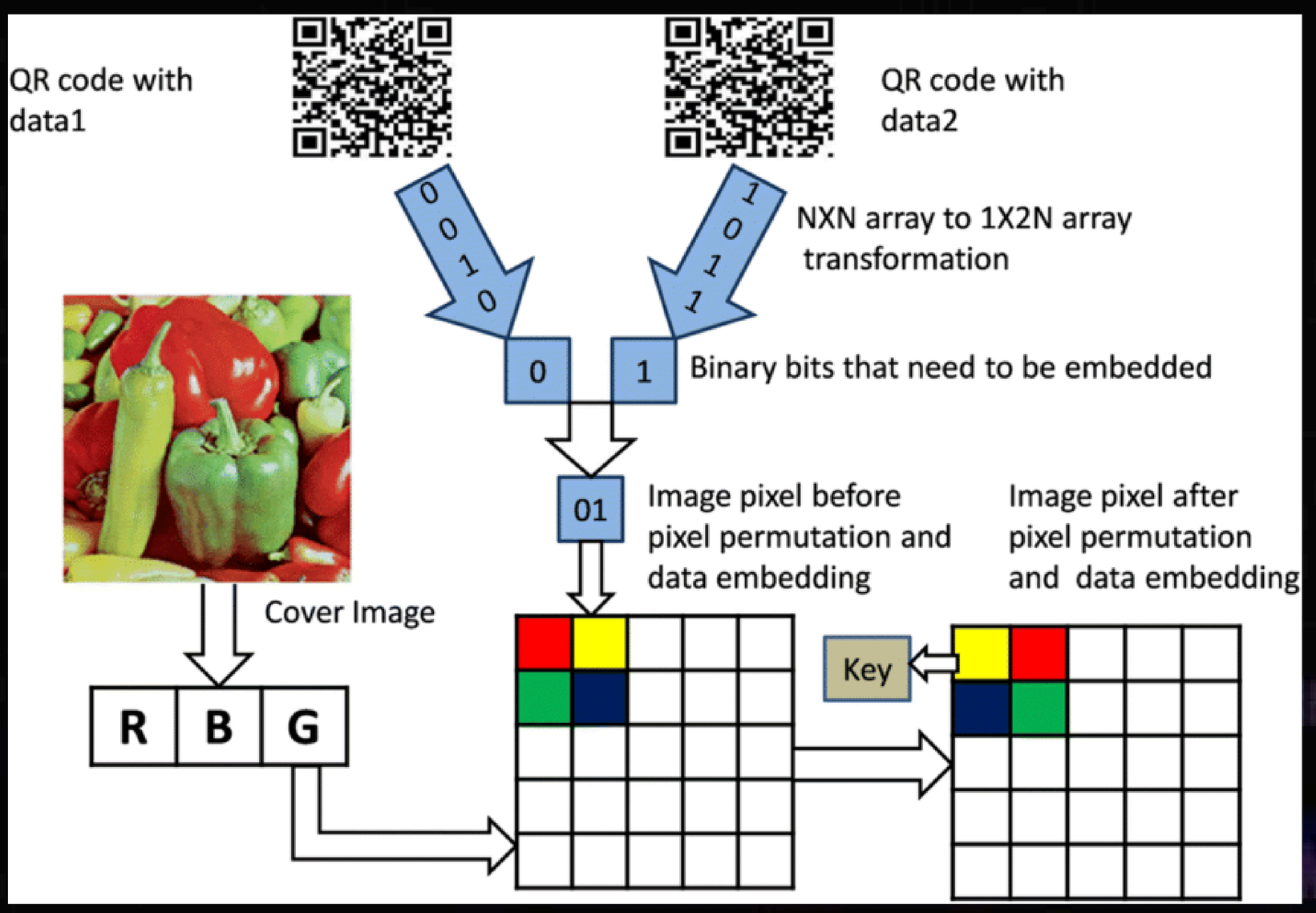
01.Pick a cover message: This can be a document, image, or anything digital that can be altered slightly.

02.Encode your secret message: Convert your message into binary or symbols that can be used for a rearrangement code.

03.Rearrange the cover message: Use a special code to shuffle the parts of the cover message to hide your secret message within.

04. Send or store the hidden message: The cover message will look normal but now secretly contains your message.

05.Decode the hidden message: The recipient uses the same code to unscramble the cover message, revealing your hidden message.



- **Advantages:**

- Security: Provides an additional layer of security through the use of a secret key for permutation.

-Less Detectable: Alterations are less noticeable due to the random rearrangement of pixels or bits.

- **Limits:**

- Complexity: Requires more complex algorithms for encoding and decoding, increasing processing time.

-Key Management: Securely sharing and managing the secret key can be challenging.

03. LSB (LEAST SIGNIFICANT BITS):

- **Main Characteristics:**

One of the most common steganography techniques, LSB involves modifying the least significant bits of pixel values in an image to embed secret data. This method exploits the fact that the human eye is unlikely to notice minor color changes.

- **Early usage:**

LSB steganography emerged in the late 20th century with the rise of digital communication. Researchers explored hiding data in digital files.

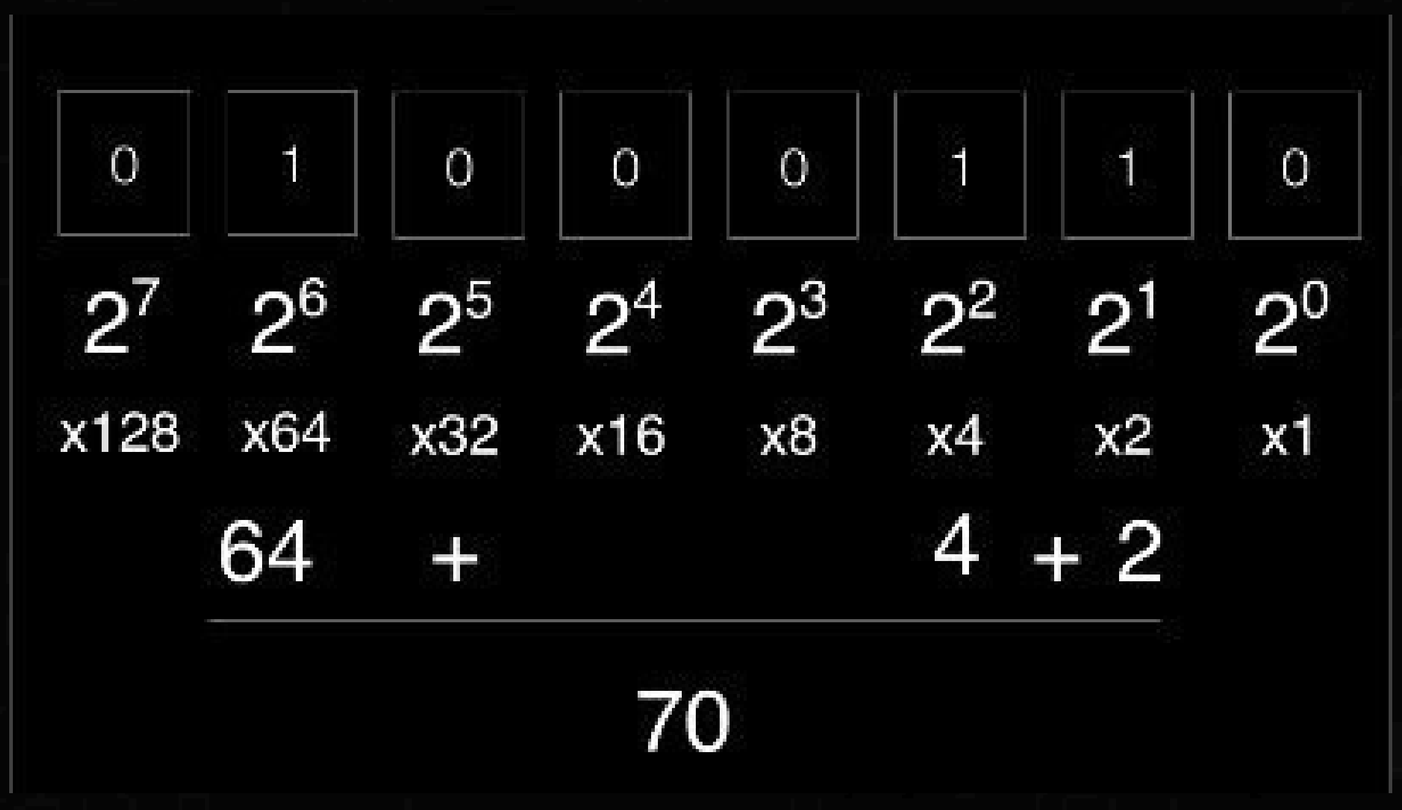
Digital images were one of the first places LSB steganography was used. Scientists found they could hide messages by changing small parts of pixels in photos that people wouldn't notice.

Text files weren't left out. Researchers looked at hiding messages in things like ASCII codes and punctuation in text documents.

Early research on LSB steganography was documented in publications about information security and computer science. These publications explained how LSB steganography worked and how it was used in digital media.

- **The binary presentation:**

In digital computing, data is represented using the binary number system, which consists of two digits: 0 and 1. Each digit in a binary number is called a "bit" (short for binary digit). Binary representation is used to encode information in a way that electronic devices, such as computers, can process and manipulate it.



<u>Bits:</u> the smallest unit of data in computing and can have a value of either 0 or 1. Multiple bits are used to represent more complex data, such as characters, numbers, or images. The number of bits needed to represent a piece of data depends on the range or size of values that need to be encoded.

- **How it works:**

<u>01.Choosing the Cover Data:</u> involves selecting a file to hide the secret message within. Common choices include digital photos where modifications can be made without being easily noticeable.

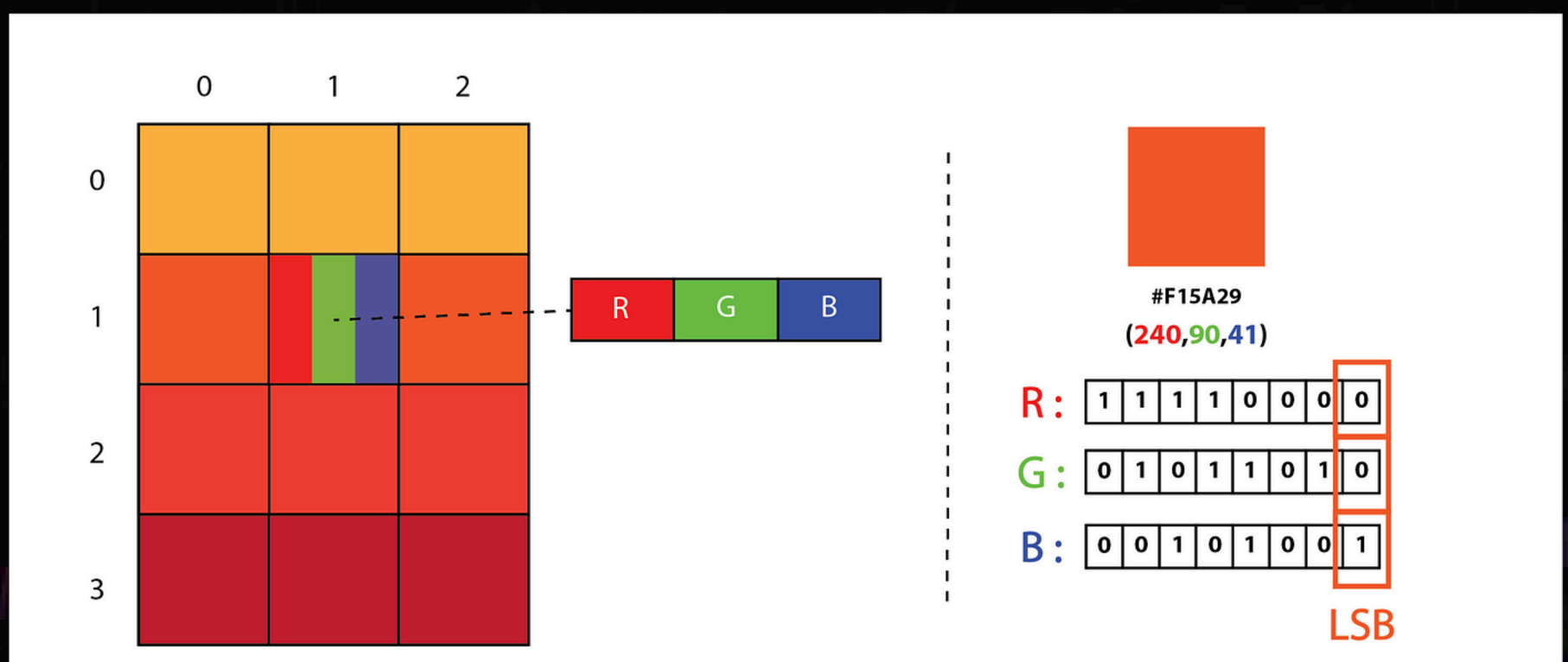
02.<u>Converting to Binary:</u> Everything in the computer world is stored as binary digits (bits). This step converts the chosen cover data (like an image) into a binary format for easier manipulation.

03.Encoding the Hidden Message: The secret message you want to hide is also converted into a binary format. This message can be text, another image, or any other data type.

04.LSB Replacement: This is where the actual hiding happens. Least Significant Bit (LSB) steganography replaces the least significant bits of the cover data with the bits from your hidden message. This is done carefully to minimize any noticeable changes to the cover data.

05.Embedding Completion: The process continues until all the bits from your hidden message are embedded into the LSBs of the cover data. Now the modified cover data contains the hidden message.

06.Transmission or Storage: The final step involves storing the modified cover data (which now has the hidden message embedded) or sending it through a communication channel. The secret message remains hidden within the LSBs of the cover data.



- **Advantages:**

- Simplicity: Relatively easy to implement and understand.

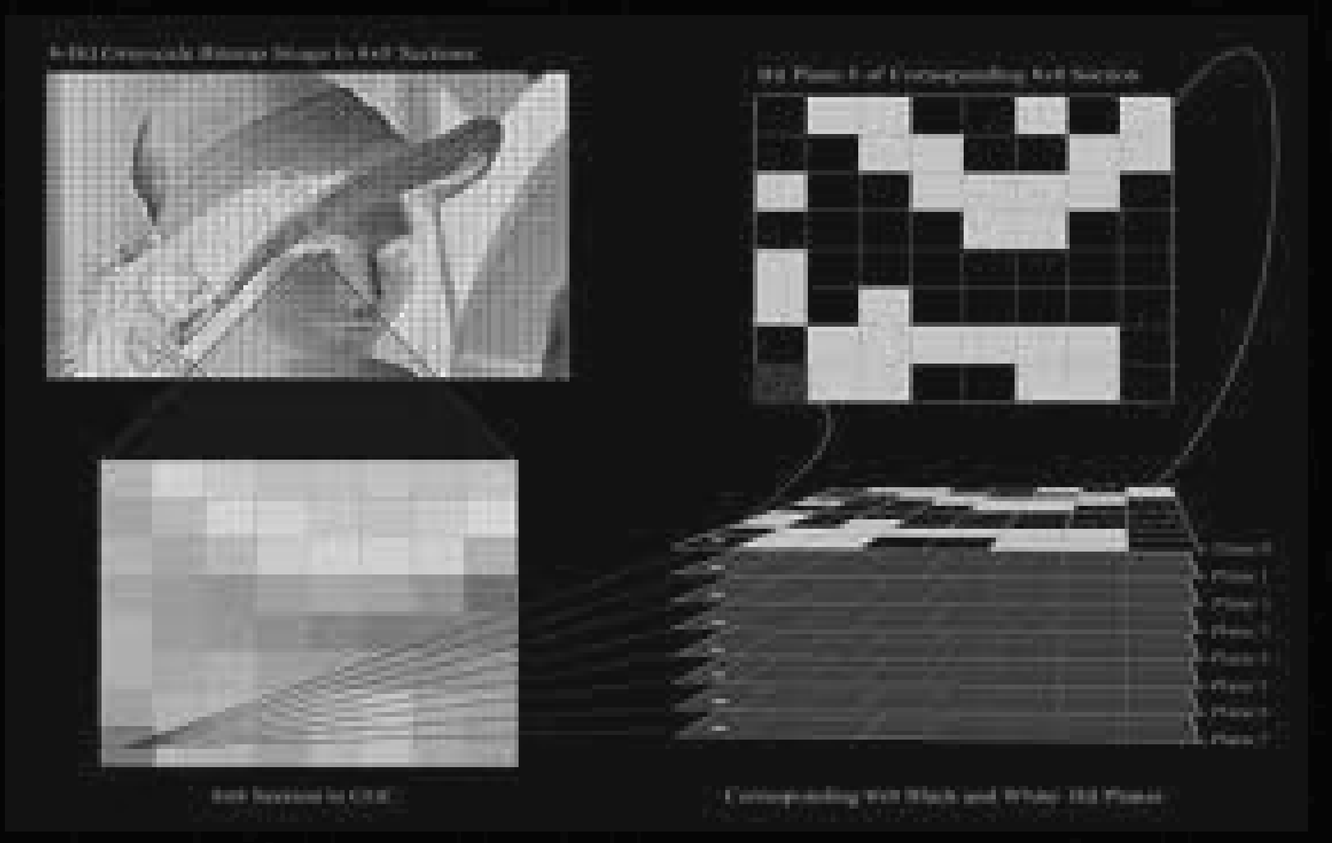
-High Capacity: Can embed a significant amount of data without noticeable changes to the image.

- **Limits:**

- Vulnerability: Techniques like statistical analysis can detect the presence of hidden data, especially in uniform color areas.

- Lossy Compression: This does not work well with formats that use lossy compression, like JPEG, as the compression process can destroy the hidden data.

04.BPCS (BIT-PLANE COMPLEXITY SEGMENTATION):



- **Main Characteristics:**

BPCS steganography exploits the complexity regions within an image's bit-planes to hide information. Complex areas, such as edges or textures, are replaced with segments of the secret data.

- **How it works:**

01.Splitting the Image: The image is broken down into its individual bit planes, which represent the binary data of each pixel.

02.Identifying Hiding Spots: Each bit plane is analyzed for complexity. Smooth (low complexity) areas are identified as good places to hide data.

03.Embedding the Message: The data to be hidden is converted to binary. This binary data is then embedded into the smooth areas of the chosen bit planes.

04.Rebuilding the Image: Finally, the modified bit planes are combined to create a new image, the stego-image, containing the hidden message. This stego-image should look very similar to the original image.

- **Advantages:**

-High Capacity: Allows for a larger volume of data to be hidden compared to many other techniques.

-Visual Imperceptibility: Changes are made in visually complex areas, making detection by human observation difficult.

- **Limits:**

- Complexity: More complex to implement, requiring algorithms to identify complex regions accurately.

- Possible Degradation: If overused, it can degrade the visual quality of the image, making the alterations more detectable.

05..CSSIS (CHAOS-BASED SPREAD SPECTRUM IMAGE STEGANOGRAPHY)

- **Main Characteristics:**

This technique combines chaos theory with spread spectrum steganography. It spreads the secret information across the frequency domain of the image in a chaotic pattern, making detection difficult.

- **Advantages:**

-Enhanced Security: The chaotic distribution of data across the image makes it hard for unauthorized entities to detect and extract the hidden information.

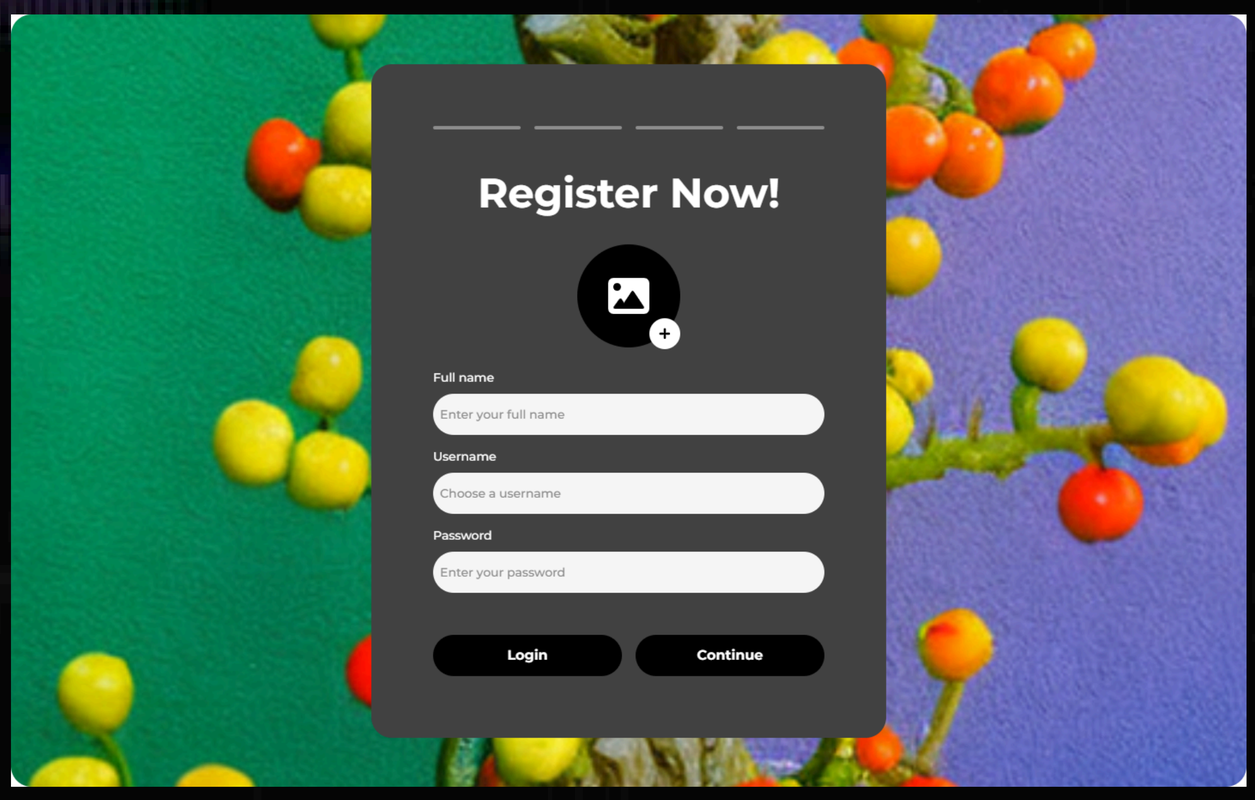
-Robustness: More resistant to image manipulations and compression compared to some other methods.

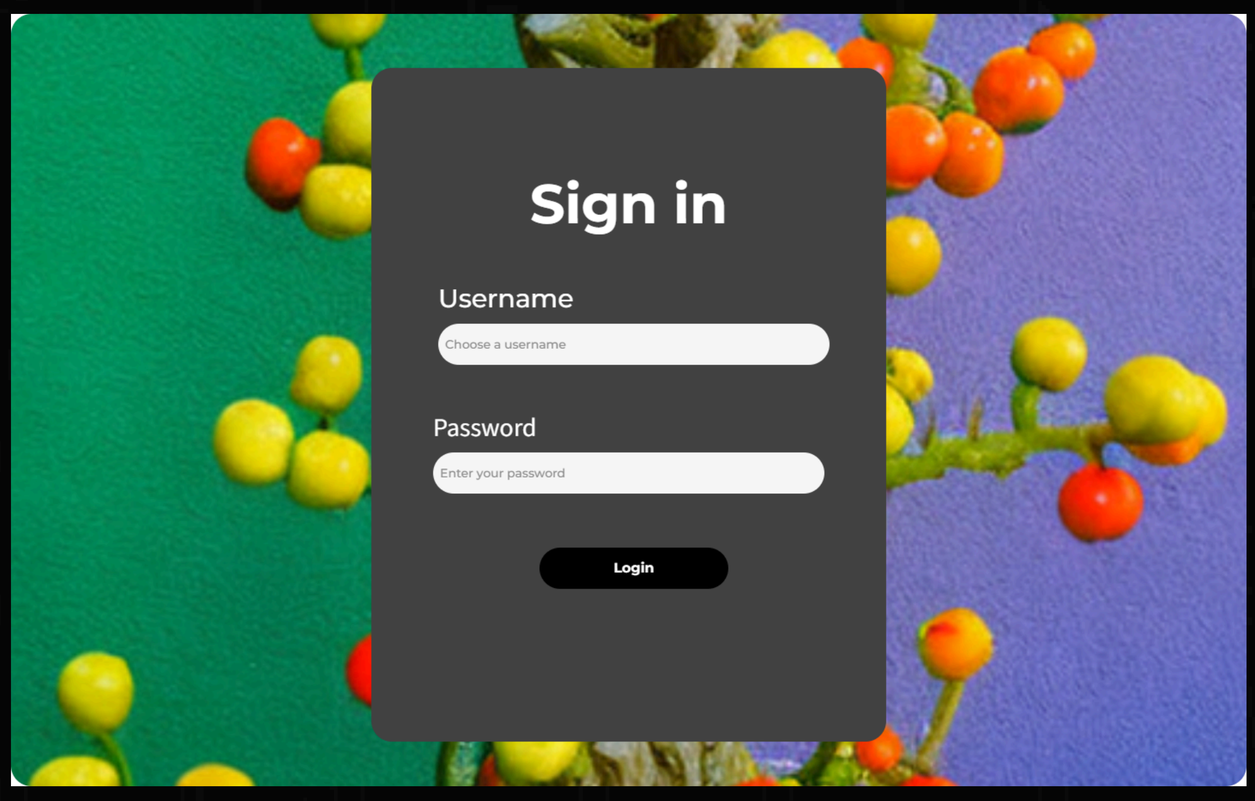
- **Limits:**

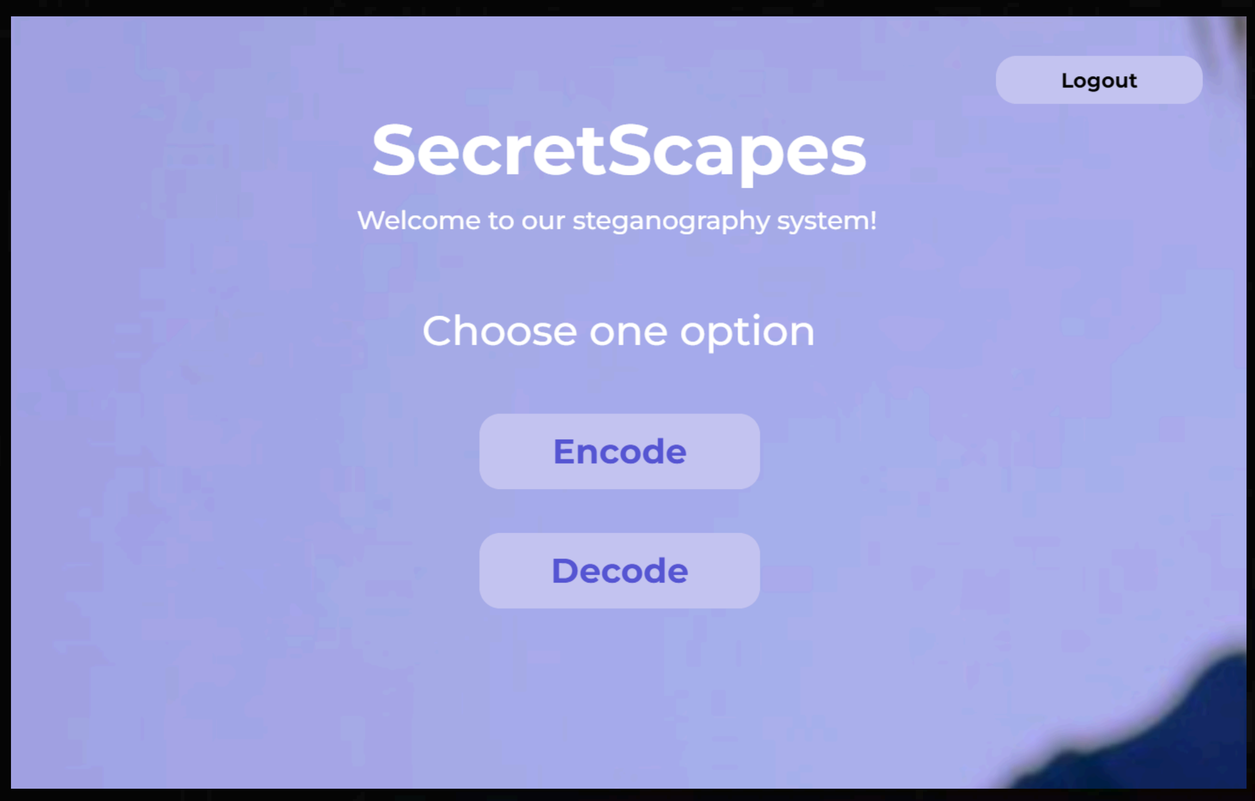
- Computational Intensity: Requires significant computational resources due to the complexity of chaotic calculations and frequency domain manipulations.

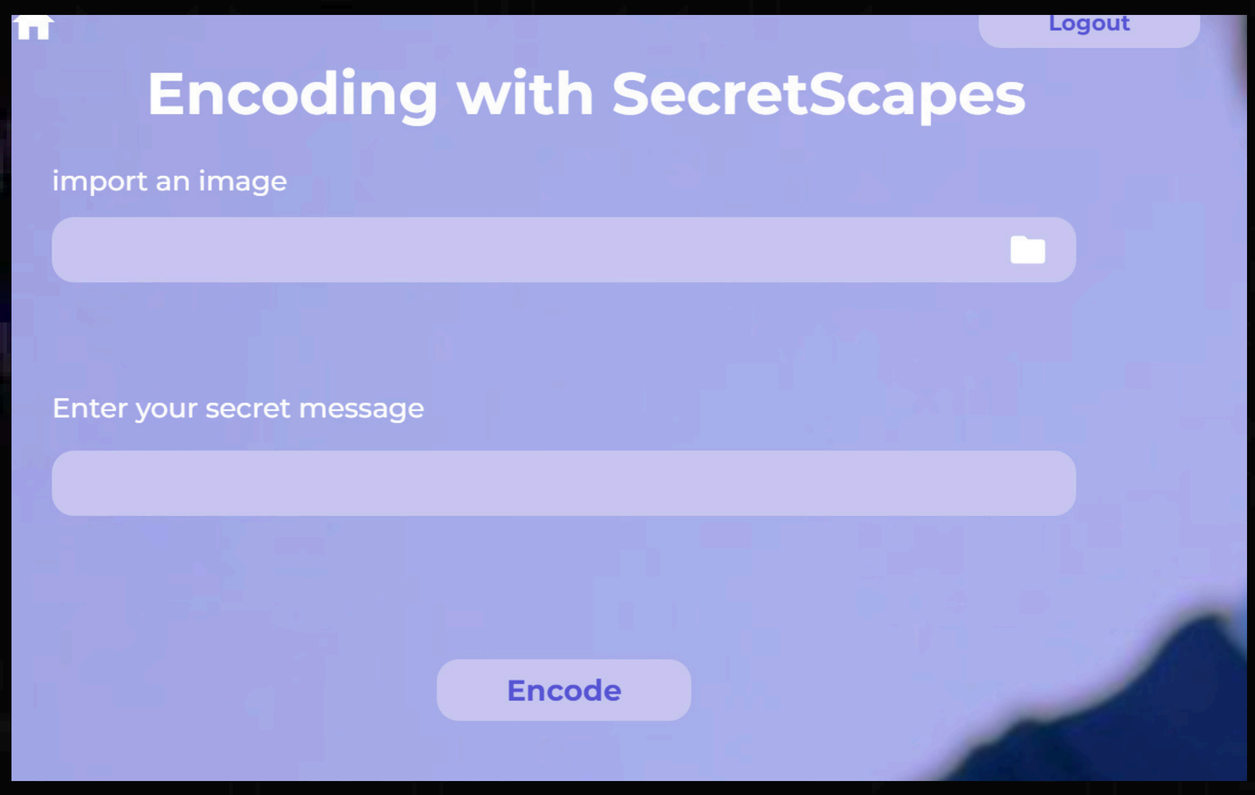
- Understanding Chaos Theory: Implementers must have a good grasp of chaos theory, adding a learning curve.

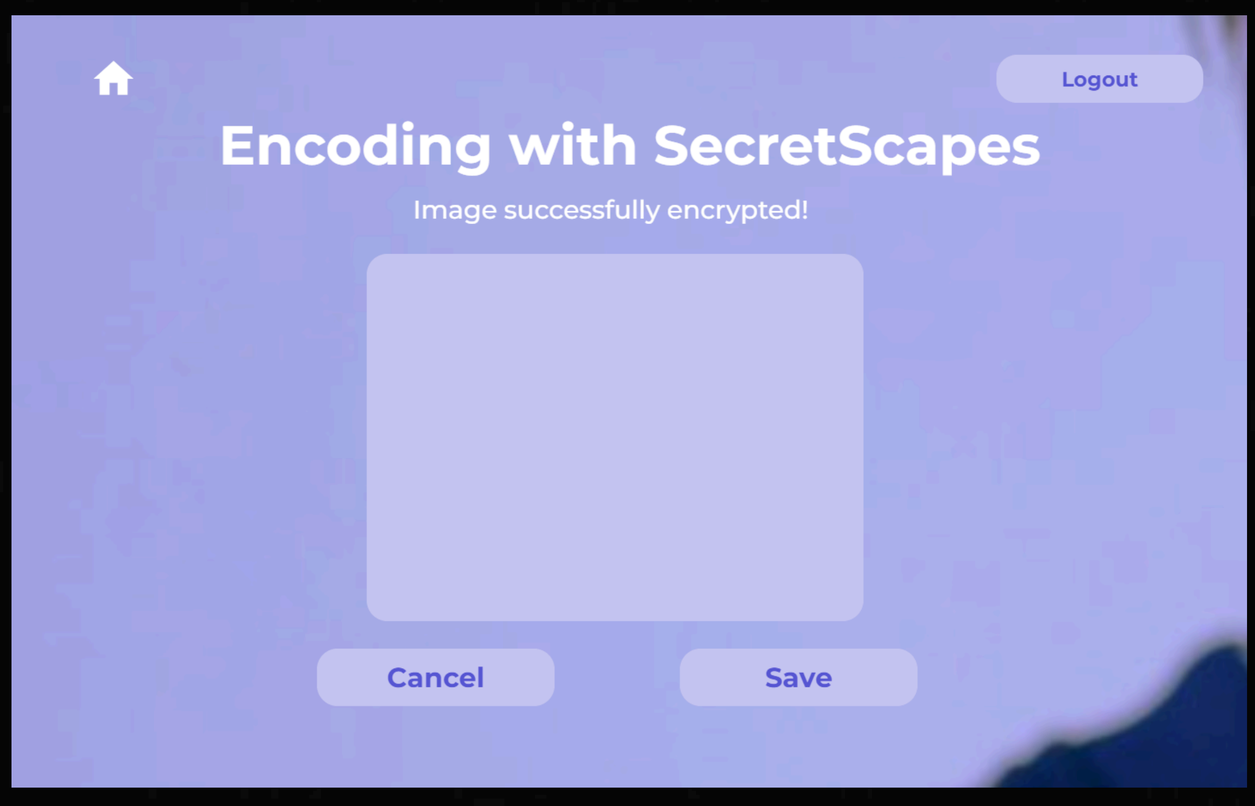
V1.0 DESIGN:

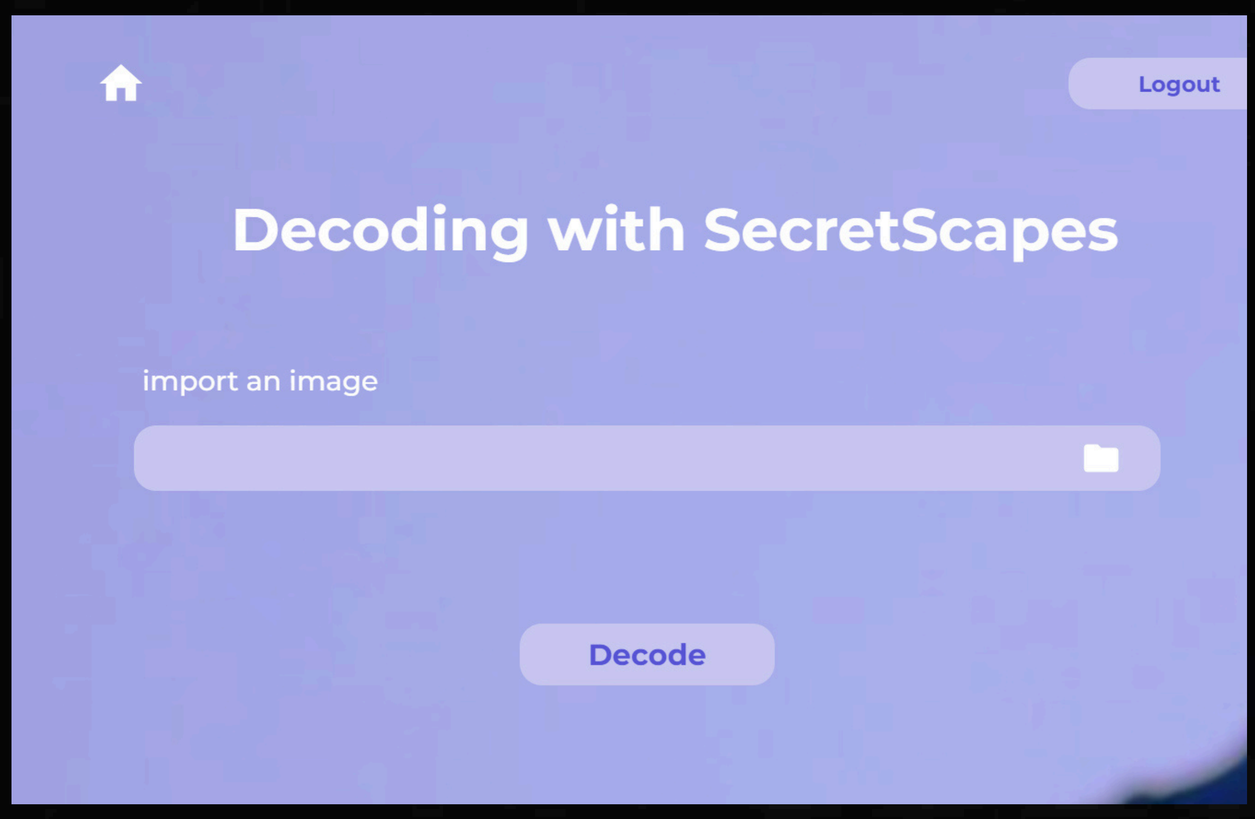

Register Now!
Full name
Enter your full name
Username
Choose a username
Password
Enter your password
Login
Continue


Sign in
Username
Choose a username
Password
Enter your password
Login

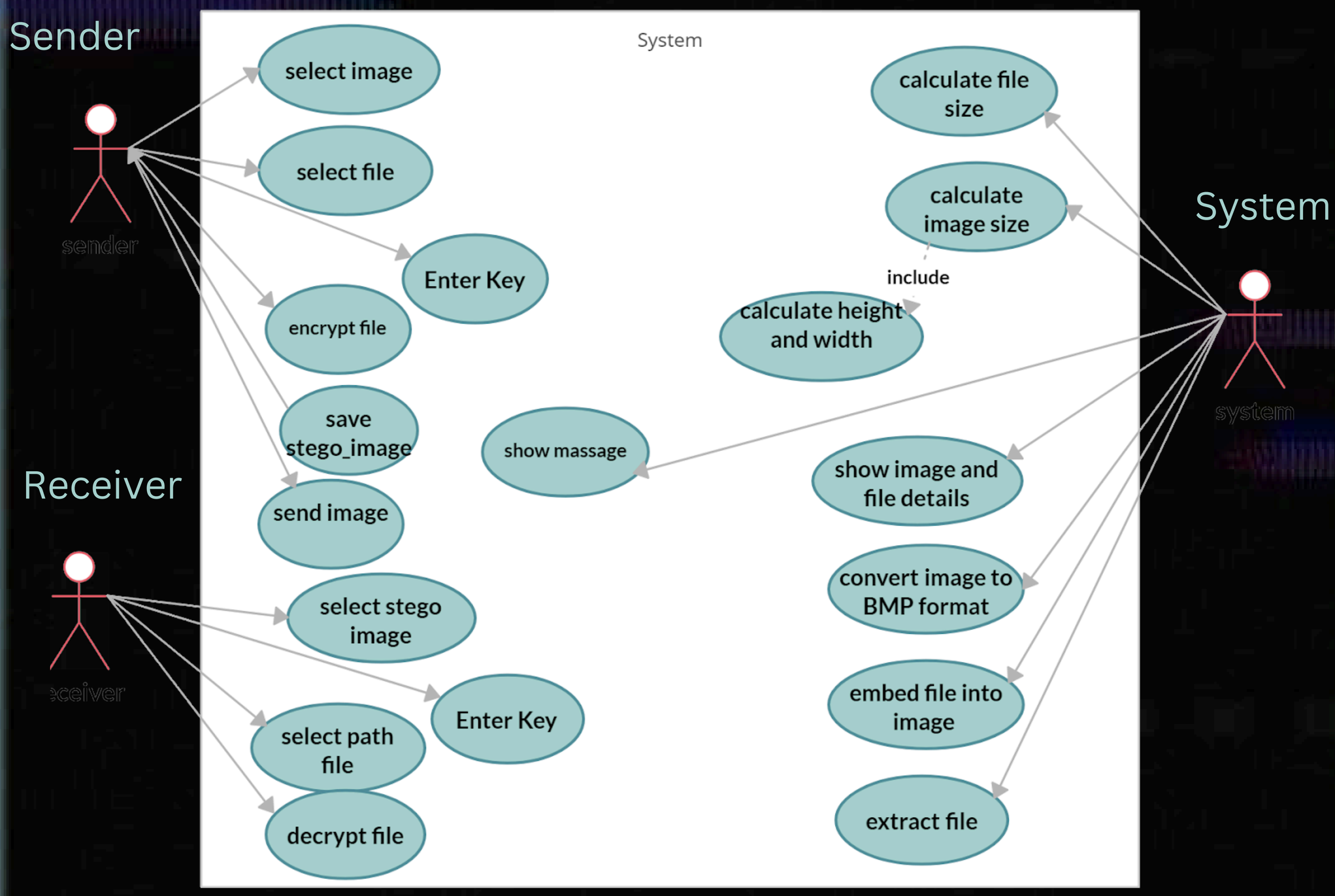

Logout
SecretScapes
Welcome to our steganography system!
Choose one option
Encode
Decode


Logout
Encoding with SecretScapes
import an image
Enter your secret message
Encode


Logout
Encoding with SecretScapes
Image successfully encrypted!
Cancel
Save


Logout
Decoding with SecretScapes
import an image
Decode

VII.USE CASE DIAGRAM:



source: https://svg.template.creately.com/iivnkyo11

VIII.EXCHANGED DATA:

- **Text Messages:**

Users can hide plaintext messages within images using our steganography technique. These messages could range from short communication snippets to longer-form textual content.

IX.USERS:

- **Sender:**

Embeds the secret message into the cover image using the steganography algorithm and shares the stego image.

- **Receiver:**

Extracts the hidden message from the stego image using the same algorithm (or the shared secret key if a more complex technique is used).

X.DESIGN OF COMPONENTS:

- **Encryption Component:**

First, the writer puts their text into the encryption part. Then, this part locks the text using a strong lock, like AES encryption. After that, the locked text is what comes out.

- **Image Component:**

Next, the writer picks an image where they want to hide the locked text. The image part takes in the chosen image and the locked text. It hides the locked text inside the image using the LSB method.

- **Decryption Component:**

Now, the user puts the hidden image into the decryption part. This part finds and takes out the locked text from the image using the LSB method. What comes out is the locked text.

- **Decryption Algorithm:**

The user gives a key to the decryption part to unlock and open the locked text.

- **Encryption Component:**

The decryption part uses a strong method like AES decryption to unlock the locked text.

XI. WORKING FLOW:

In an image steganography website, the sender begins by accessing the platform and inputting the secret message they wish to conceal. This message is then encrypted using a secure encryption algorithm, such as AES encryption, generating an encoded version of the text. Meanwhile, the sender selects an image from their device to serve as the carrier for the hidden message. The encoded message is then embedded into the selected image using a steganographic technique, such as the Least Significant Bit (LSB) method, ensuring that the alterations to the image are imperceptible to the naked eye. Once the embedding process is complete, the sender shares the steganographic image with the receiver. Upon receiving the steganographic image, the receiver accesses the platform and uploads the image. The website's decoding mechanism extracts the hidden message from the image using the same steganographic technique employed during embedding. The extracted message is then decrypted using the appropriate decryption key provided by the sender, revealing the original plaintext message. This message is displayed to the receiver, completing the process of covert communication through image steganography.

XII. STEPS:

- **Encoding Message:**

The sender starts by encoding a confidential message, which could be text they want to keep hidden.

To enhance security, the message is often encrypted using robust encryption algorithms like AES. This ensures that even if the steganographic image is intercepted, the hidden message remains secure and unreadable without the decryption key.

- **Embedding Process:**

The encoded message is then embedded into the pixels of a chosen carrier image. The carrier image acts as a cover for the hidden message.

Various steganographic techniques are employed to embed the message while minimizing any noticeable changes to the carrier image. Techniques such as Least Significant Bit (LSB) insertion are commonly used, where the least significant bits of pixel values are replaced with bits of the hidden message.

- **Transmission and Storage:**

Once the message is embedded, the steganographic image containing the hidden data can be transmitted through communication channels or stored in digital repositories. Since the carrier image appears unchanged to the naked eye, the hidden message remains undetected to casual observers or automated systems.

- **Decoding Process:**

Upon receiving the steganographic image, the recipient initiates the decoding process to extract the hidden message. Specialized software or algorithms are used to analyze the carrier image and extract the embedded message without altering the image's visual appearance.

- **Decryption:**

If encryption was applied before embedding, the extracted message is decrypted using the appropriate decryption key. Decryption ensures that the original plaintext message is recovered accurately, allowing the recipient to access the concealed information.

XIII. PROJECT TOOLS:

- **Open CV (cv2):**

A Python library that offers a vast array of functions and algorithms for image and video processing, object detection, camera calibration, augmented reality, and more. It can used to subtly modify an image's pixel values, like the least significant bits, to embed a secret message.

- **Numpy (np):**

Modifying the least significant bits (LSBs) of pixel values in a NumPy array can encode a message without significantly altering the image's visual quality. Additionally, NumPy's mathematical functions can be used for tasks like encryption and decryption of the hidden data.

- **Types:**

Images are typically stored as arrays of numerical values representing pixel intensities (e.g., grayscale values from 0 to 255). These values are usually integers or floating-point numbers.

- **Pillow:(Python Imaging Library - PIL Fork):**

A powerful Python library for image processing and manipulation. It can be used to implement steganographic techniques by directly manipulating pixel values in images and to read, modify, and save images, which is essential for embedding and extracting hidden data.

- **Stegano:**

A Python module that provides tools for steganography, the practice of concealing a message within another message or a digital image. Stegano uses the least significant bit (LSB) technique for hiding data in images.

- **pycryptodome**

provides cryptographic functionalities is widely used for encryption, decryption, digital signatures, hashing, and various other cryptographic operations in Python applications.

XIV. PROJECT PHASES

01.:FRONTEND DEVELOPMENT

After crafting the user interface (UI) design, the next step involves translating it into code using HTML and CSS. This process transforms the visual representation of the UI into a set of structured elements and styles that web browsers can interpret and render.

01.Encryption:

```
<div>
    <section>
        <div class="logout-button">
            <a href="Signin.html"><button id="logout">Logout</button></a>
          </div>
          <div class="home-button1">
            <a href="Selection.html"><button id="logout"><i class="fa-solid fa-house"></i></button></a>
        </div>
        <h1 class="inti">SecretScapes</h1> <br>
  <h2>Import an image</h2> <br>
  <input type="file" id="imageInput" accept="image/*"><br>
   </section>
    <section>
         <h2>Enter your secret message</h2><br>
        <textarea placeholder="Enter your secret message"></textarea><br> <br><br>
    </section>
    <br><button type="button">Encode</button><br>
</div>
  </main>
 </body>
```

02.Decryption:

```
<div>
   <section>
        <div class="logout-button">
            <a href="Signin.html"><button id="logout">Logout</button></a>
          </div>
          <div class="home-button1">
           <a href="Selection.html"><button id="logout"><i class="fa-solid fa-house"></i></button></a>
       </div>
        <h1 class="inti">SecretScapes</h1> <br>
 <h2>Import an image</h2> <br>
 <input type="file" id="imageInput" accept="image/*"><br>
  </section>
   <br><button type="button">Decode</button><br>
</div>
```

02.:AES ALGORITHM

01.Libraries import:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad
from Crypto.Util.Padding import unpad
```

02.Encryption Function:

```
def encrypt_message(message):
    key = get_random_bytes(16)  # Generate a random 128-bit key
    cipher = AES.new(key, AES.MODE_ECB)
    padded_message = pad(message.encode(), AES.block_size)
    ciphertext = cipher.encrypt(padded_message)
    return key, ciphertext
```

03.Decryption Function:

```
def decrypt_message(encrypted_message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_message = cipher.decrypt(encrypted_message)
    # Unpad the decrypted message
    decrypted_message = unpad(decrypted_message, AES.block_size)
    return decrypted_message.decode()
```

04.Example:

```
# Example usage:
message = "Hello, world!"
key, encrypted_message = encrypt_message(message)
print("Key:", key)
print("Encrypted message:", encrypted_message)
decrypted_message = decrypt_message(encrypted_message, key)
print("Decrypted message:", decrypted_message)
```

03.LSB ALGORITHM

01. Convert the message to a binary one:

```
def messageToBinary(msg):
  if type(msg) == str:
    return ''.join([ format(ord(i), "08b") for i in msg ])
  elif type(msg) == bytes or type(message) == np.ndarray:
    return [ format(i, "08b") for i in msg ]
  elif type(msg) == int or type(msg) == np.uint8:
    return format(msg, "08b")
  else:
    raise TypeError("Input type not supported")
```

02.Hide the data (a function that will return an image) & Show the data (a function that will return the decoded data)

03. Encode the data into the image and decode it

```
# Encode data into image
def encode_text():
  image_name = input("Enter image name(with extension): ")
  image = cv2.imread(image_name) # Read the input image using OpenCV-Python.
  #It is a library of Python bindings designed to solve computer vision problems.

  data = input("Enter data to be encoded : ")
  if (len(data) == 0):
    raise ValueError('Data is empty')

  filename = input("Enter the name of new encoded image(with extension): ")
  encoded_image = hideData(image, data) # call the hideData function to hide the
  cv2.imwrite(filename, encoded_image)
```

```
# Decode the data in the image
def decode_text():
  # read the image that contains the hidden image
  image_name = input("Enter the name of the steganographed image that you want to
  image = cv2.imread(image_name) #read the image using cv2.imread()

  text = showData(image)
  return text
```

04.Mainloop (Steganography function)

```
# Image Steganography
def Steganography():
    a = input("Image Steganography \n 1. Encode the data \n 2. Decode the data
    userinput = int(a)
    if (userinput == 1):
      print("\nEncoding....")
      encode_text()

    elif (userinput == 2):
      print("\nDecoding....")
      print("Decoded message is " + decode_text())
    else:
        raise Exception("Enter correct input")

Steganography() #encode image
```

04.BCPS ALGORITHM:

01.Encoding:

```
def bcps_encode(image_path, secret_message, output_path):

    img = Image.open(image_path)
    width, height = img.size
    secret_message_binary = ''.join(format(ord(char), '08b') for char in secret_messag
    img_gray = img.convert('L')
    img_array = np.array(img_gray)
    bit_planes = [np.bitwise_and(img_array, 2**i) for i in range(8)]
    complexities = [np.mean(np.abs(np.diff(plane))) for plane in bit_planes]
    threshold = sum(complexities) / len(complexities)
    encoded_bit_planes = []
    for i, plane in enumerate(bit_planes):
        if complexities[i] > threshold:
            encoded_plane = np.where(plane > 0, plane | 1, plane)
        else:
            encoded_plane = plane
        encoded_bit_planes.append(encoded_plane)
    encoded_img_array = sum(encoded_bit_planes)
    encoded_img = Image.fromarray(encoded_img_array.astype('uint8'))
    encoded_img.save(output_path)
```

02.Decoding:

```python
def bcps_decode(image_path):

    encoded_img = Image.open(image_path)
    encoded_img_array = np.array(encoded_img)

    bit_planes = [np.bitwise_and(encoded_img_array, 2**i) for i in range(8)]

    complexities = [np.mean(np.abs(np.diff(plane))) for plane in bit_planes]

    threshold = sum(complexities) / len(complexities)

    extracted_bits = [plane & 1 for index, plane in enumerate(bit_planes) if complexit

    extracted_message_binary = ''
    for i in range(encoded_img_array.shape[0]):
        for j in range(encoded_img_array.shape[1]):
            extracted_bit = 0
            for plane in extracted_bits:
                extracted_bit |= plane[i, j]
            extracted_message_binary += str(extracted_bit)
            if len(extracted_message_binary) % 8 == 0:
                break
```

XV. GITHUB LINK:

All the provided details are shared in ImageSteganographySystem repository in github.

Link:

https://github.com/rayenrouaissi/ImageSteganographySystem