

StegaX

IMAGE STEGANOGRAPHY PROJECT REPORT

The ultimate guide to the art and science of hiding secret messages

IT360: Information Assurance & Security

Prepared by:

Mohamed Aziz Srairi (BA/IT)

Imen Cherif (BA/IT)

Rayen Fares (BA/IT)

Takwa Karaoud (Fin/IT)

Abstract

Image steganography is the technique of hiding secret information within an image to ensure confidentiality and secure communication.

This project aims to develop an image steganography system that allows users to hide text messages within digital images.

The system utilizes various steganographic techniques to embed the secret and encrypted message in the image while maintaining the visual quality of the cover image.

The steganography operation will be implemented through an interactive user interface that will enable the user to login, choose to encode or decode a message, pick an image, and finally send the hidden message to the receiver.

This report provides an overview of the project, its objectives, implementation details, and evaluation results.

Table of Content:

I.	Introduction	4
II.	Literature Review:	5
III.	Functional Requirements	30
IV.	Non-Functional Requirements.....	31
V.	Objectives of the Proposed Solution:	32
VI.	Design of the components.....	33
VII.	Methodology	40
VIII.	Project Tools:.....	41
IX.	Project Phases:	43
X.	Implementation Details:	45
XI.	Evaluation:.....	46
XII.	Conclusion:	47
XIII.	Future Work:	48
XIV.	References	49

I. Introduction

- **Purpose:**

With the development of digital systems and the rise of cyber communication, secure communication channels have become of utmost importance. This can be ensured through the use of cryptography, or steganography, which is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message. In particular, image steganography involves hiding a secret message within a digital image. The image can then be transmitted through any communication channel, with the secret message only being able to be extracted by the intended recipient.

- **Scope:**

The image steganography system described in this document will allow users to hide secret messages within digital images and extract them later. The system will provide a user-friendly interface to allow the system's users to enter their messages, upload the images that will hide their secrets, and extract the messages from the steganographic images. The system will ensure the implementation of various security measures in order to prevent unauthorized access to the messages.

II. Literature Review:

1. Theoretical Review:

The Image Steganography System is a software application that allows users to hide secret messages within digital images using various steganography techniques. The system will be based on the Python programming language, providing a simple user interface for encoding and decoding messages from uploaded image files.

○ Definitions :

- **Steganography:** the technique of hiding secret data within an ordinary, non secret, file or message in order to avoid detection; the secret data is then extracted at its destination.
- **Digital image:** A digital image is a representation of a real image as a set of numbers that can be stored and handled by a digital computer. In order to translate the image into numbers, it is divided into small areas called pixels
- **Encryption:** the method by which information is converted into secret code that hides the information's true meaning.
- **Decryption:** a process that transforms encrypted information into its original format.

• Other Types of Steganography:

- **Audio Steganography:** This technique hides information within audio files by altering the sound waves in a way that is imperceptible to the human ear. This can be achieved by adding noise to the least significant bits of the audio samples or by changing the amplitude or frequency of the audio signal. The hidden data can be extracted by analyzing the audio file using special software.
- **Video Steganography:** This type of steganography embeds data within video files by altering the frames or using motion vectors. For example, individual frames could be modified slightly to represent one bit of the hidden data, which can be extracted by analyzing the video file using special software.
- **Text Steganography:** This technique hides information within plain text by using various methods such as letter frequency analysis, word substitution, or embedding the data within the text using steganographic codes. For example, a message could be hidden in the spaces between words or in the first letter of each word.
- **Network Steganography:** This technique hides data within network packets by manipulating unused header fields or by encoding data within seemingly innocuous traffic. For example, data could be hidden within the padding bytes of a packet or by using a protocol that is designed to look like legitimate traffic.

2. Empirical Review:

a. Product Perspective :

The main components of image steganography are as follows:

- **Secret Message:** The secret message is the data that the sender wants to be hidden within the image. The secret message can be in the form of text, audio, video, or any other type of digital data.
- **Cover Image:** The image in which the secret message is hidden is called the cover image. The cover image can be any digital image format such as a JPEG, PNG, or BMP file.
- **Steganography Algorithm:** The steganography algorithm is a set of rules and procedures that are used to hide the secret message within the cover image. There are several steganography algorithms that will be explained in the next parts of the document such as LSB, BCPS, CSSIS, etc.
- **Steganography Key:** The steganography key is a password that will be used to encrypt the secret message before it is hidden within the cover image. The steganography key ensures that only authorized users can access the hidden message.

b. Product Functions :

The functional flow of image steganography can be explained in the following steps:

- **Input:** The user chooses the cover image and writes the secret message.
- **Encryption:** The secret message is encrypted using a steganography key.
- **Embedding:** The steganography algorithm is used to hide the encrypted secret message within the cover image.
- **Output:** The modified cover image is produced with the encrypted secret message embedded within it.

- **Decryption:** The recipient of the cover image uses the steganography key to decrypt the hidden message.
- **Output:** The decrypted secret message is produced as output.

c. User Characteristics :

The image steganography system is designed for users who need a secure communication channel for transmitting sensitive information. The intended users of the system include individuals, organizations, and businesses that need to ensure that their messages are transmitted securely and cannot be intercepted by unauthorized third parties.

d. General Constraints :

The system will require Python 3.8 installed on the device, along with the following libraries: OpenCV, NumPy, Tkinter, and Pillow. Other libraries and software may also be added.

e. Assumptions and Dependencies: Acceptance Criteria :

- **Encoding and Decoding :**

The system should be able to encode and decode messages from digital images using the specified steganography technique. The system should be able to encode and decode messages without altering the quality of the original image.

- **User Interface:**

The system should provide a user-friendly interface that allows users to interact with the system easily.

- **Performance:**

The system should be able to encode and decode messages from images quickly and efficiently, but also handle large image files and messages without slowing down or crashing.

- **Security:**

The security of the Image Steganography System is of utmost importance. The system should ensure that encoded messages are not detectable and that they cannot be easily deciphered by unauthorized users.

The following security requirements must be met:

- **Message Encryption:**

The system should provide an encryption functionality to ensure that messages are encoded with strong encryption algorithms before being hidden in images. This will make it more difficult for unauthorized users to decipher the message.

- **Password Protection:**

The system should provide password protection functionality to prevent unauthorized access to encoded messages. The system should require users to provide a password before they can access the encoded message.

- **User Authentication:**

The system should provide user authentication functionality to ensure that only authorized users can access the system. Users should be required to enter a username and password to access the system.

- **Steganography Techniques:**

The system should offer multiple steganography techniques that are difficult to detect. The system should not rely on a single steganography technique as it can be more easily detected and decrypted.

- **Image Metadata Protection:**

The system should remove all metadata information from images used for encoding messages, to prevent unauthorized users from accessing information that could reveal the existence of a hidden message.

f. Steganography Algorithms:

There are several algorithms that can be used for image steganography. Some of the popular algorithms are:

i. Least Significant Bit Algorithm

General Introduction:

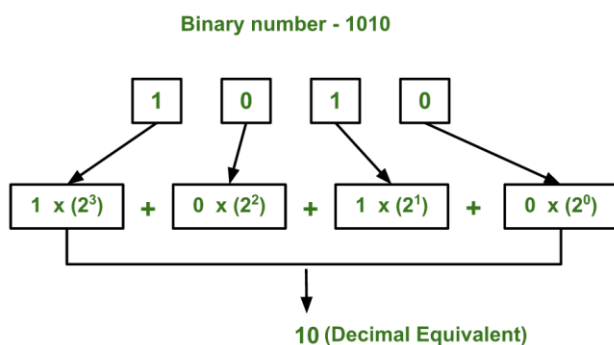
The Least Significant Bit (LSB) algorithm is a **steganography method** that uses the least significant bits of pixels or audio samples to conceal information within an image or audio file. The least significant bits (LSB) algorithm takes advantage of the fact that the human eye is not sensitive to minute variations in color or brightness to allow us to embed secret data without degrading the quality of the cover image or audio file.

Early usage:

In the early days of steganography, the LSB algorithm was first applied, and it is still widely utilized today. As digital communication technologies became more common and advanced in the 1990s, it is known that the LSB algorithm attracted more attention and research. It is a quick and easy way to conceal information in an image or audio file without degrading the quality of the output. More advanced steganography techniques have been created to avoid detection, but it is also fairly simple to do so.

Binary representation and bits:

In digital systems, each binary digit (or bit) represents a power of 2. The rightmost (or least significant) bit represents 2^0 (or 1), the second rightmost bit represents 2^1 (or 2), the third rightmost bit represents 2^2 (or 4), and so on.



Decimal To Binary Conversion:

Let the decimal number be : 14

2	14	0
2	7	1
2	3	1
2	1	1
	0	

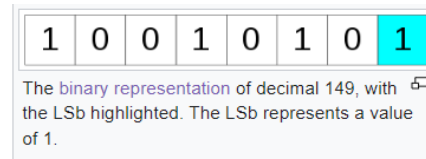
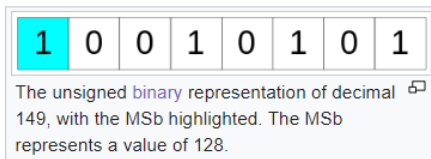
$(14)_{10} = (1110)_2$

Let the decimal number be : 22

2	22	0
2	11	1
2	5	1
2	2	0
2	1	1
	0	

$(22)_{10} = (10110)_2$

The leftmost non-zero bit with the largest positional value is known as Most Significant Bit (MSB) whereas the rightmost bit with the smallest positional value is known as Least Significant Bit (LSB)

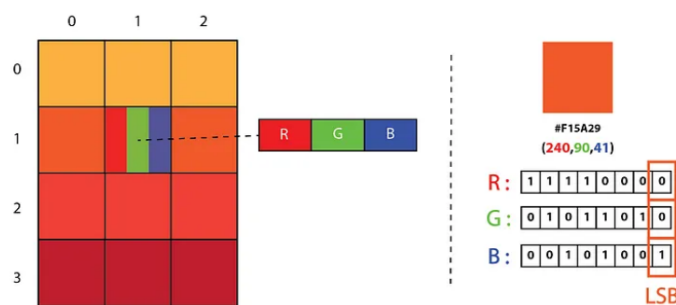


How it actually works:

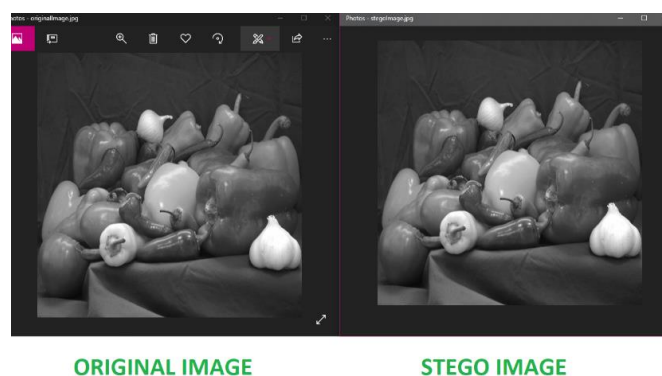
How are the bits manipulated?

In the context of the LSB algorithm, the least significant bit of a pixel refers to the rightmost (or least significant) bit of the binary representation of the pixel value. For example, if the binary representation of a pixel is 01100101, the least significant bit is 1.

By replacing the least significant bit of each pixel with a bit from the message to be hidden, the LSB algorithm changes the binary representation of the pixel value only slightly. This means that the modified image will look very similar to the original image, and the hidden message will not be easily visible to the naked eye.



Representation of Image as a 2D Array of RGB Pixels



Strengths and Weaknesses :

- **Strengths:**

Simplicity: The LSB algorithm is relatively simple to implement and can be used for hiding small amounts of data in digital images.

Imperceptibility: The changes made by the LSB algorithm are usually imperceptible to the human eye. This means that the image can be used for communication without arousing suspicion.

Robustness: The LSB algorithm is robust against common image processing techniques such as cropping, scaling, and compression. This means that the hidden information can be retrieved even after the image has been processed.

- **Weaknesses:**

Vulnerability to detection: The LSB algorithm is vulnerable to detection by statistical analysis techniques. This is because the algorithm introduces a pattern in the image that can be detected using statistical analysis techniques.

Low capacity: The LSB algorithm has low capacity for hiding data in digital images. This is because only one bit can be hidden in each pixel of the image.

Susceptibility to attacks: The LSB algorithm is susceptible to various attacks such as noise addition and LSB replacement. These attacks can be used to destroy or modify the hidden information in the image.

Possible enhancements :

Improve the capacity: One way to improve the capacity of the LSB algorithm is to use higher-order bits for data hiding. For example, instead of using only the least significant bit of each pixel, one can use the two or three least significant bits. This will increase the capacity of the algorithm and enable it to hide more data in the image.

Use Encryption: Before embedding the message in the image, one can use encryption techniques to scramble it, increasing the LSB algorithm's susceptibility to statistical analysis methods. This will make it more challenging for an attacker to find the image's hidden information.

Use distortion measures: Distortion measures can be used to evaluate the quality of the stego-image and ensure that the changes made to the image are imperceptible. This can help to improve the imperceptibility of the algorithm and make it more difficult for an attacker to detect the hidden information.

What does the algorithm do :

The encoding is done using the following steps:

1. Convert the image to grayscale
2. Resize the image if needed
3. Convert the message to its binary format
4. Initialize the output image the same as the input image
5. Traverse through each pixel of the image and do the following:
 - Convert the pixel value to binary
 - Get the next bit of the message to be embedded
 - Create a variable **temp**
 - If the message bit and the LSB of the pixel are the same, set temp = 0
 - If the message bit and the LSB of the pixel are different, set temp = 1
 - This setting of temp can be done by taking the XOR of the message bit and the LSB of the pixel
 - Update the pixel of the output image to input image pixel value + **temp**
6. Keep updating the output image till all the bits in the message are embedded
7. Finally, write the input as well as the output image to the local system.

Code snippet :

```
from PIL import Image

def hide_message(image_path, message):

    # Open the image and convert it to grayscale
    image = Image.open(image_path).convert('L')

    # Convert the message to binary
    binary_message = ''.join(format(ord(c), '08b') for c in message)

    # Get the size of the image
    width, height = image.size
```

```

# Check if the message can fit in the image
if len(binary_message) > width * height:
    raise ValueError("Message too large to fit in the image.")

# Iterate over each pixel in the image
for x in range(width):
    for y in range(height):
        # Get the pixel value (a number from 0 to 255)
        pixel = image.getpixel((x, y))

        # Get the binary representation of the pixel value
        binary_pixel = format(pixel, '08b')

        # Check if there is still message to embed
        if len(binary_message) > 0:
            # Replace the least significant bit of the pixel with the next bit of the message
            new_pixel = int(binary_pixel[:-1] + binary_message[0], 2)
            binary_message = binary_message[1:]
        else:
            # No more message to embed, keep the pixel as is
            new_pixel = pixel

        # Set the new pixel value
        image.putpixel((x, y), new_pixel)

# Save the image with the hidden message
image.save(image_path)

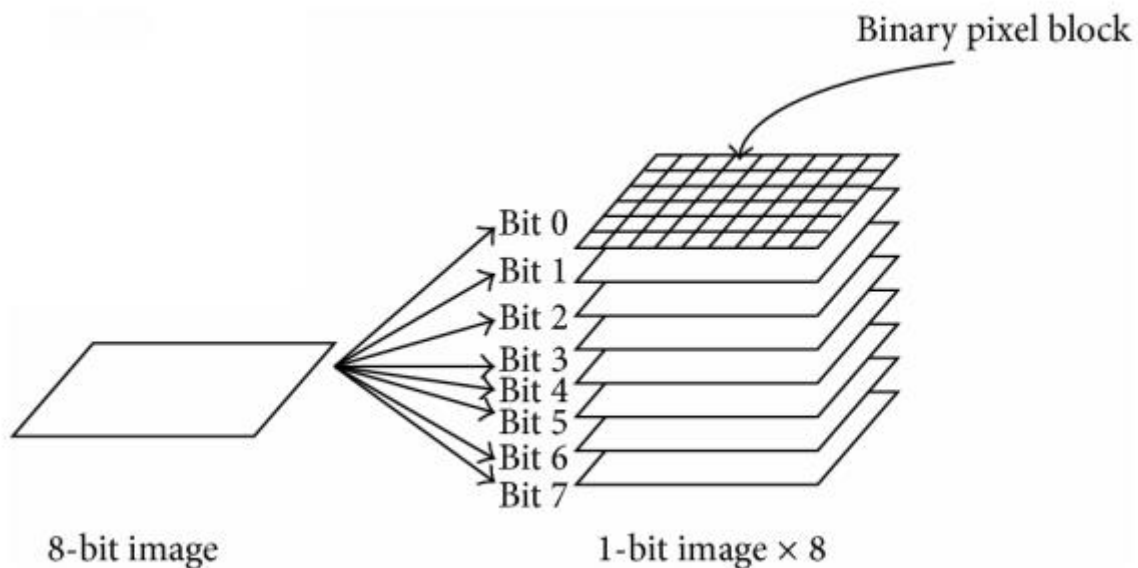
# Example usage
hide_message('image.png', 'Hello, world!')

```

ii. Bit plane complexity segmentation algorithm (BPCS):

As previously mentioned, the least significant bits (LSB) substitution is the most well-known spatial steganographic system. Although this method is simple and easy, it is weak in robustness and compression, such as JPEG compression.

BPCS steganography was first put forward by Kawaguchi and Eason. The basic principle is that firstly the cover image is divided into “informative region” and “noise-like region.” Then the secret information is hidden in noise-like blocks of cover image. In LSB technique, data is hidden in the lowest bit-plane. But in BPCS technique, data is hidden in pixel blocks of all the planes, from the highest plane (most significant bit, MSB plane) to the lowest plane (LSB plane), which have noisy patterns. In BPCS, a gray image consisting of n -bit pixels can be decomposed into n -binary planes. For example, $n=8$, as shown below.



Overall View:

The bit plane complexity segmentation algorithm is an image segmentation technique that is based on the concept of bit planes. In this algorithm, an image is represented as a matrix of pixels, each of which has a binary representation. The bit plane complexity segmentation algorithm works by dividing an image into multiple segments based on the complexity of the image's bit planes.

The basic steps involved in the bit plane complexity segmentation algorithm are as follows:

1. Convert the input image to grayscale.
2. Divide the grayscale image into bit planes, where each bit plane represents a different bit of the pixel values.
3. Calculate the complexity of each bit plane by counting the number of transitions between 0 and 1 within each bit plane.

4. Divide the image into segments based on the complexity of the bit planes. For example, you could group together pixels whose bit planes have a complexity within a certain range.

Main Characteristics:

The bit plane complexity segmentation algorithm is a method for segmenting images based on the complexity of the image's bit planes. Here are some of its main characteristics:

1. Bit plane decomposition: The algorithm decomposes the image into its bit planes, which represent different levels of detail in the image. The bit planes with higher values represent more complex details, while the bit planes with lower values represent simpler details.
2. Complexity thresholding: The algorithm uses a threshold value to determine which bit planes to include in the segmentation. The threshold value is chosen based on the characteristics of the image being segmented, and can be adjusted to produce different levels of segmentation complexity.
3. Edge preservation: The algorithm is able to preserve edges and details in the image, even in areas with low contrast. This is because it segments the image based on the complexity of the bit planes, which often corresponds to the presence of edges.
4. Robustness to noise: The bit plane complexity segmentation algorithm is robust to noise and can handle images with varying levels of noise. This is because it uses the bit planes of the image, which are less sensitive to noise than the original image.
5. Flexibility: The algorithm is flexible and can be adapted to a wide range of image processing and computer vision tasks. The parameters of the algorithm, such as the complexity threshold, can be adjusted to suit the specific characteristics of the image being segmented.
6. Interpretability: The algorithm produces segmentations based on the complexity of the bit planes, which are easy to interpret and can provide insights into the structure and content of the image.

Uses of the BPCS Algorithm:

The bit plane complexity segmentation algorithm can be used in a variety of image processing and computer vision applications, for example:

1. Object recognition: The bit plane complexity segmentation algorithm can be used to segment objects in an image based on their complexity, which can help in identifying and recognizing objects.

In fact, it can be used in object recognition as a preprocessing step for segmenting an image into regions of interest. By segmenting an image based on the complexity of its bit planes, BPCS can effectively capture the edges and texture information in an image, which are important features for object recognition.

In object recognition using BPCS, the segmented regions can be further processed to extract features such as texture, color, and shape. These features can then be used to train a machine learning algorithm to recognize objects in the image.

BPCS has been shown to be effective in object recognition tasks, particularly in scenarios where the objects of interest have well-defined boundaries and textures. However, it may not perform as well in more complex scenarios with cluttered backgrounds or overlapping objects, where other segmentation algorithms may be more suitable.

2. Medical imaging: The algorithm can be used for segmentation of medical images to identify different structures within the images, such as tumors, blood vessels, and organs.

BPCS can be effective in medical imaging applications because it can capture the subtle differences in texture and contrast that are important for identifying regions of interest in medical images. For example, BPCS has been used for the segmentation of brain tumors in MRI images, where the texture and contrast differences between the tumor and healthy tissue are often subtle.

In addition to segmenting images, BPCS can also be used for feature extraction in medical image analysis. The segmented regions can be further processed to extract features such as texture, shape, and intensity, which can then be used for diagnosis, treatment planning, and disease tracking.

Overall, BPCS is a promising technique for medical imaging applications, but its effectiveness depends on the specific application and the quality of the input image.

3. Remote sensing: The bit plane complexity segmentation algorithm can be used in satellite imagery to segment images into land cover types, such as forest, water, and urban areas.

Remote sensing refers to the process of gathering information about the Earth's surface using sensors mounted on satellites or aircraft. This information can be used for a variety of purposes such as land use classification, environmental monitoring, and disaster response.

In remote sensing, BPCS can be used for image segmentation based on the complexity of the bit planes, which captures important texture and contrast information in the image. This can be useful for identifying different land cover types such as forests, water bodies, and agricultural fields.

4. Video compression: The algorithm can be used in video compression techniques to identify regions of an image that are less complex and therefore require less bits to represent.

In video compression, BPCS can be used for segmenting video frames based on the complexity of their bit planes. The segmented regions can then be encoded separately, with the more complex regions using a higher bit rate and the less complex regions using a lower bit rate. This can improve the efficiency of compression by allocating more bits to regions of the video that contain more important visual information.

BPCS can also be used for reducing the amount of data that needs to be transmitted or stored for video compression. By segmenting the video frames into regions of interest, only the most important regions need to be transmitted or stored at high quality, while the less important regions can be compressed more aggressively.

5. Security and surveillance: The bit plane complexity segmentation algorithm can be used in security and surveillance applications to detect and track moving objects based on their complexity.

In security and surveillance, BPCS can be used for object detection and tracking by segmenting the image or video frames into regions of interest based on the complexity of the bit planes. This can be useful for identifying and tracking moving objects such as vehicles or people, as well as for detecting changes in the scene such as the presence of new objects or changes in the environment.

BPCS can also be used for image and video encryption in security and surveillance applications. By segmenting the image or video frames into regions of interest based on their complexity, different regions can be encrypted with different levels of security. For example, highly complex regions can be encrypted with stronger encryption algorithms to ensure their security, while less complex regions can be encrypted with weaker algorithms to save processing time.

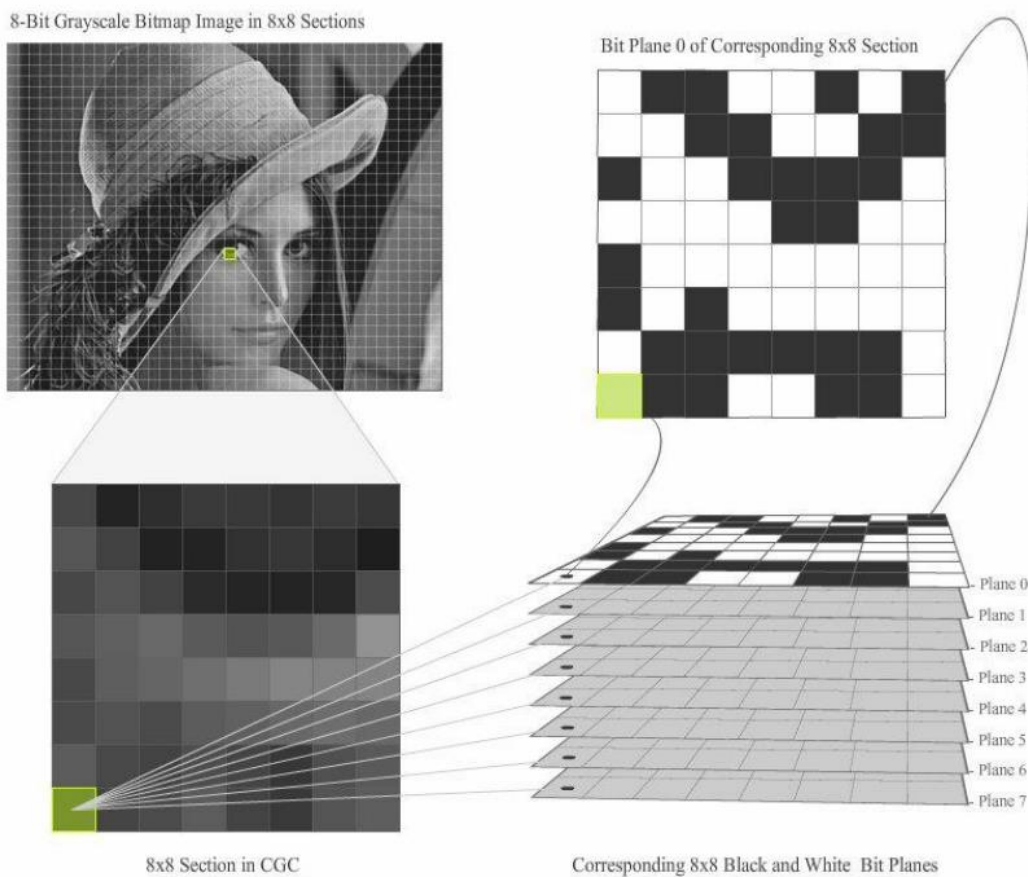
6. Aerospace: BPCS (Bit-Plane Complexity Segmentation) can also be used in aerospace applications, particularly in the analysis of satellite imagery for various purposes such as land use classification, crop monitoring, and disaster response.

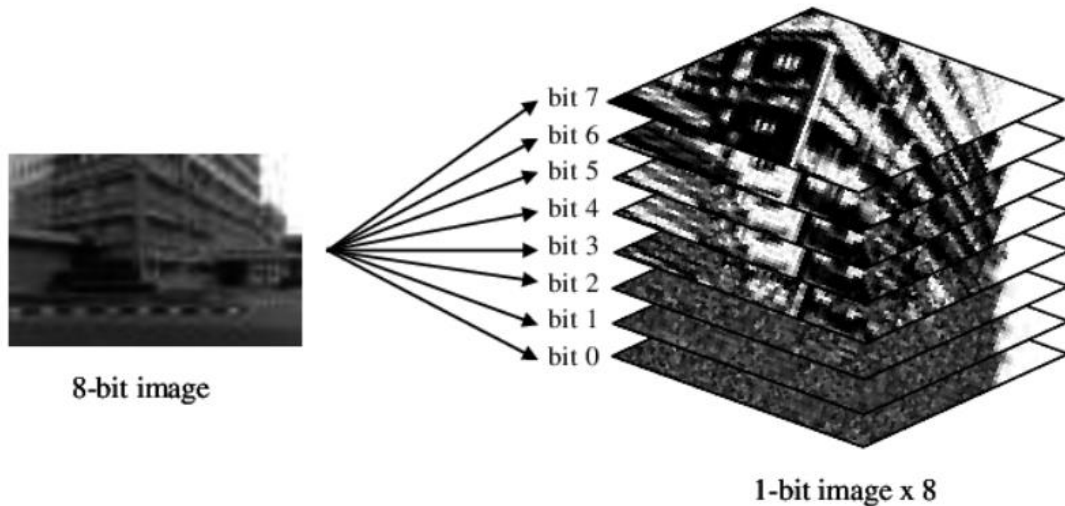
In satellite imagery, BPCS can be used for segmentation of different land cover types based on their texture and contrast. For example, BPCS can be used to differentiate between urban and rural areas based on the presence of buildings, roads, and vegetation. It can also be used for identifying specific crop types and monitoring their growth and health.

In addition, BPCS can be useful for disaster response by identifying damaged or destroyed infrastructure and buildings. For example, after a natural disaster such as an earthquake or flood, BPCS can be used to identify areas where buildings have collapsed or roads have been destroyed.

The BPCS Algorithm Process:

1. The input image is decomposed into its bit planes. Each bit plane represents a different level of detail in the image, from the most significant bit (MSB) to the least significant bit (LSB).
2. The complexity of each bit plane is calculated using a complexity metric, such as the edge density or texture complexity. This metric is used to determine the threshold value for the bit planes.
3. The bit planes above the threshold value are combined to create a binary mask. This mask represents the areas of the image that are more complex and likely to contain object boundaries or edges.
4. The binary mask is used to segment the original image using a segmentation algorithm, such as a region growing algorithm or a watershed algorithm.





BPCS steganography. A cover 8-bit image is decomposed into 8 binary images prior to the embedding process.

BPCS in Python:

```

1  import cv2
2  import numpy as np
3
4  # Load the image
5  img = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
6
7  # Calculate the bit planes
8  bit_planes = [np.uint8((img >> i) & 1) * 255 for i in range(8)]
9
10 # Calculate the complexity of each bit plane
11 complexity = [np.sum(np.abs(cv2.Sobel(plane, cv2.CV_64F, 1, 0, ksize=3))) + np.sum(np.abs(cv2.Sobel(plane, cv2.CV_64F, 0, 1, ksize=3)))
12               for plane in bit_planes]
13
14 # Calculate the threshold value
15 threshold = np.mean(complexity)
16
17 # Combine the bit planes above the threshold
18 binary_mask = np.zeros_like(img)
19 for i in range(8):
20     if complexity[i] > threshold:
21         binary_mask += bit_planes[i]
22
23 # Apply image segmentation
24 ret, segmented_image = cv2.threshold(binary_mask, 0, 255, cv2.THRESH_BINARY)
25
26 # Display the results
27 cv2.imshow('Input Image', img)
28 cv2.imshow('Segmented Image', segmented_image)
29 cv2.waitKey(0)
30 cv2.destroyAllWindows()
31

```

Strengths of the BPCS Algorithm:

The bit plane complexity segmentation algorithm has several advantages over other image segmentation techniques. One of the main advantages is its ability to preserve edges and details in an image, even in areas with low contrast. It can also handle images with varying lighting conditions and noise levels.

In fact, its main strengths are:

1. **Edge preservation:** The algorithm is able to preserve edges and details in an image, even in areas with low contrast. This is because it segments the image based on the complexity of the bit planes, which often corresponds to the presence of edges.
2. **Robustness to noise:** The bit plane complexity segmentation algorithm is robust to noise and can handle images with varying levels of noise. This is because it uses the bit planes of the image, which are less sensitive to noise than the original image.
3. **Parameterization:** The algorithm allows for flexible parameterization, which makes it adaptable to a wide range of image segmentation tasks. The parameters can be tuned to suit the specific characteristics of the image being segmented.
4. **Application versatility:** The bit plane complexity segmentation algorithm can be applied to a wide range of image processing and computer vision tasks, including object recognition, medical imaging, remote sensing, video compression, and security and surveillance.
5. **Interpretability:** The algorithm produces segmentations based on the complexity of the bit planes, which are easy to interpret and can provide insights into the structure and content of the image.

Weaknesses of the BPCS Algorithm:

Like any image segmentation algorithm, the bit plane complexity segmentation algorithm has some weaknesses that should be considered when choosing a segmentation method. Here are some of its main weaknesses:

1. **Computationally intensive:** The algorithm can be computationally intensive and may require a large amount of memory, especially when working with high-resolution images. This can make it impractical for real-time or large-scale applications.

2. **Parameter selection:** The algorithm requires careful selection of the parameters used to segment the image, such as the threshold for the bit plane complexity. These parameters can be difficult to choose and may require manual tuning for each image.
3. **Sensitivity to illumination:** The bit plane complexity segmentation algorithm can be sensitive to changes in illumination, which can affect the complexity of the bit planes and lead to inaccurate segmentations.
4. **Limited segmentation accuracy:** While the algorithm can preserve edges and details in an image, it may not provide the same level of accuracy as more complex segmentation methods. This can lead to under-segmentation or over-segmentation in certain cases.
5. **Limited applicability to color images:** The bit plane complexity segmentation algorithm is typically applied to grayscale images, and may not be suitable for color images without additional preprocessing or modification.

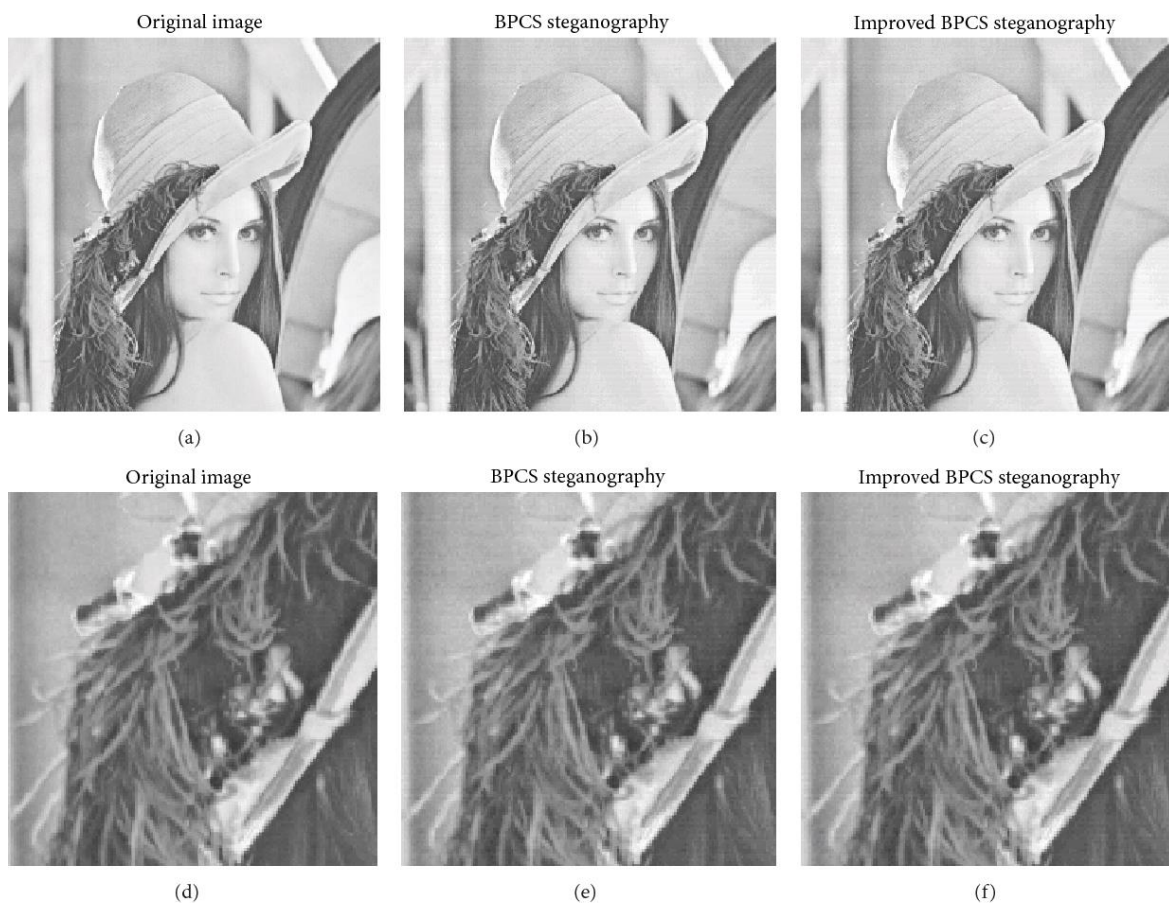
How can the BPCS Algorithm be improved?

There are several ways in which the bit plane complexity segmentation algorithm can be improved. Here are some suggestions:

1. **Speed optimizations:** To improve the speed of the algorithm, researchers can explore techniques such as parallelization or optimization of data structures and algorithms.
2. **Parameter optimization:** Researchers can develop automated methods for selecting the optimal parameters for the algorithm, such as the threshold for the bit plane complexity. This could involve machine learning techniques or statistical analysis of the image properties.
3. **Color image support:** Researchers can extend the algorithm to support color images. This could involve developing new techniques for representing and segmenting color information in the bit planes.
4. **Illumination normalization:** To reduce the sensitivity of the algorithm to illumination changes, researchers can explore techniques for normalizing the illumination in the image before segmentation, such as histogram equalization or adaptive thresholding.

5. Integration with other segmentation methods: The bit plane complexity segmentation algorithm can be combined with other segmentation methods to improve accuracy and speed. For example, the algorithm could be used to provide an initial segmentation, which could then be refined using other techniques such as clustering or region growing.

Overall, the bit plane complexity segmentation algorithm has a lot of potential for improvement, and research in this area is ongoing. By addressing some of the weaknesses of the algorithm, researchers can make it even more useful for a wide range of image processing and computer vision tasks.



iii. Chaos-based Spread Spectrum Image Steganography (CSSIS) Algorithm

Introduction :

Chaos-based Spread Spectrum Image Steganography (CSSIS) is an algorithm used for image steganography. The CSSIS algorithm is an extension of the traditional Spread Spectrum Image Steganography (SSIS) algorithm, which is based on the concept of embedding a secret message in the frequency domain of an image that uses **chaos theory*** and **spread spectrum communication*** to hide secret information in an image. It was proposed in 2004 by J. Fridrich, M. Goljan, and R. Du.

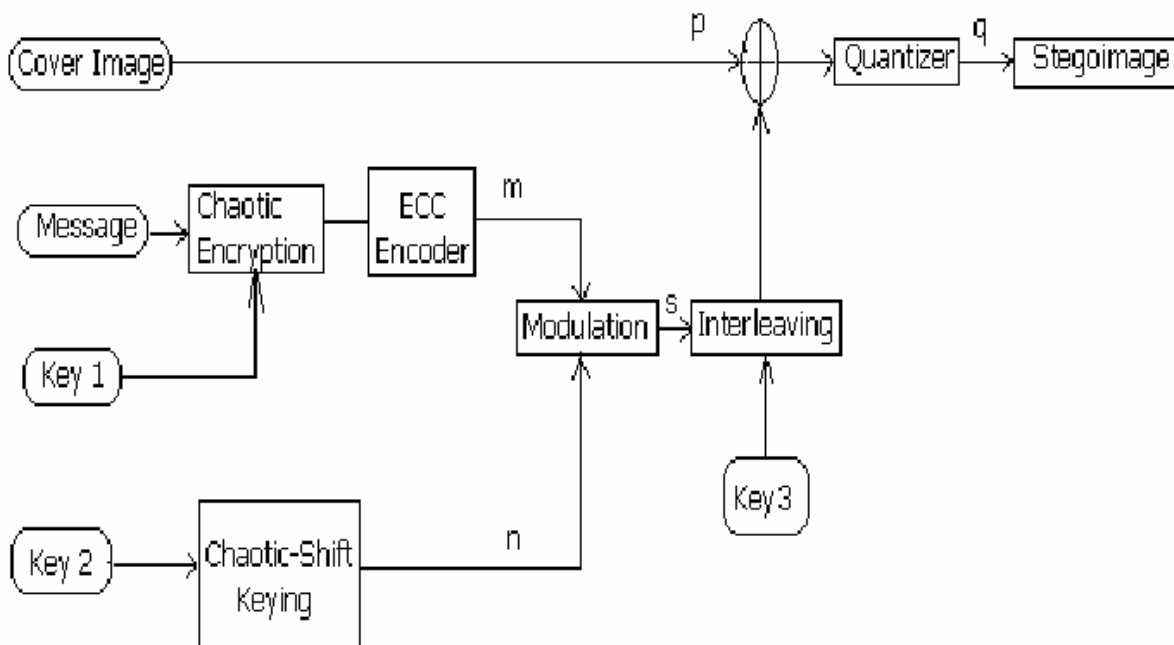


Fig. 1. CSSIS encoder.

Chaos theory is the study of dynamical systems that are highly sensitive to initial conditions. Even small changes in the initial state of a chaotic system can lead to vastly different outcomes. Some key characteristics of chaotic systems include:

Determinism: Chaotic systems are deterministic, meaning they have no random elements. Their behavior is fully determined by their initial conditions. However, the deterministic nature can lead to unpredictable behavior.

Sensitivity to initial conditions: Chaotic systems are highly sensitive to tiny changes in initial conditions. Even an infinitesimally small change can lead to significantly different behavior. This is commonly known as the "butterfly effect".

Non-linearity: Chaotic systems are non-linear, meaning there is no proportionality between inputs and outputs. The relationship is complex.

Aperiodicity: Chaotic systems never repeat or fully settle into a steady state. Their behavior appears random and disorderly.

Boundedness: Although chaotic systems appear random, their behavior is limited to a bounded range. The values do not explode to infinity.

Some examples of natural chaotic systems include weather, ocean currents, heart rhythms, and population growth. Chaos theory has applications in various fields like mathematics, physics, engineering, economics, biology, and philosophy.

****Spread spectrum communication*** is a type of wireless communication that uses a wide range of frequencies to transmit data. It is used to reduce interference and improve signal strength. In spread spectrum communication, the data is spread across multiple frequencies, which makes it more difficult for an eavesdropper to intercept the data. Additionally, spread spectrum communication is used to reduce the power required to transmit data, and to increase the range of the signal.

How it works:

The CSSIS algorithm works by dividing the original image into non-overlapping blocks of equal size. Each block is then transformed using a chaos-based function, which introduces a degree of randomness and non-linearity into the transformation process. The transformed blocks are then represented by a single integer value, which is the sum of the color indices of all the pixels in that block. The secret message is then converted into binary format, and each binary bit of the secret message is embedded in the least significant bit of the block value for a specific block. The modified block values are then used to construct the steganographic image. *The recipient can extract the hidden message by reversing the CSSIS algorithm.*

Steps:

1. The original image is divided into non-overlapping blocks of equal size.
2. Each block is transformed using a chaos-based function, which introduces a degree of randomness and non-linearity into the transformation process.
3. The transformed blocks are then represented by a single integer value, which is the sum of the color indices of all the pixels in that block.
4. The secret message is then converted into binary format.
5. Each binary bit of the secret message is embedded in the least significant bit of the block value for a specific block.
6. The modified block values are then used to construct the steganographic image.

Strengths & Weaknesses:

Strengths:

1. Good security. The use of chaos and spread spectrum makes the technique robust to attacks like noise addition, filtering, and compression.

2. High embedding capacity. Since the signal is spread, it can achieve a high data hiding rate.
3. The algorithm can be used with any digital image format, making it a flexible and versatile technique.

Weaknesses:

1. Requires the exchange of system parameters. To extract the message, the receiver needs to know details about the chaotic system used.
2. May introduce some distortion to the image. Spreading the signal can modify the image slightly.
3. CSSIS may increase the size of the carrier image, making it difficult to transmit large amounts of data.
4. The computational complexity of the algorithm may slow down the transmission process.
5. The algorithm may be vulnerable to attacks if the key used for encoding and decoding the messages is compromised.

Code Snippet:

```
import numpy as np
from PIL import Image

def encode(image_path, message, key):
    # Load the image
    image = Image.open(image_path)
    # Convert the image to a numpy array
    image_array = np.array(image)
    # Generate a pseudo-random sequence using chaos theory
    sequence = generate_sequence(key, len(message))
    # Convert the message to binary
    binary_message = ''.join(format(ord(i), '08b') for i in message)
    # Embed the message in the image using spread spectrum technique
    index = 0
```

```

for i in range(len(image_array)):
    for j in range(len(image_array[0])):
        for k in range(len(image_array[0][0])):
            if index < len(binary_message):
                image_array[i][j][k] =
int('{0:b}'.format(image_array[i][j][k]).zfill(8)[-2] +
binary_message[index] + binary_message[index + 1], 2) ^ sequence[index //
2]

                index += 2

# Save the modified image
modified_image = Image.fromarray(image_array)
modified_image.save('modified_' + image_path)
def decode(image_path, key):
    # Load the image
    image = Image.open(image_path)
    # Convert the image to a numpy array
    image_array = np.array(image)
    # Generate the same pseudo-random sequence used for encoding
    sequence = generate_sequence(key, len(binary_message))
    # Extract the hidden message from the image using spread spectrum
technique
    binary_message = ''
    index = 0
    for i in range(len(image_array)):
        for j in range(len(image_array[0])):
            for k in range(len(image_array[0][0])):
                if index < len(sequence):
                    binary_message +=
str((int('{0:b}'.format(image_array[i][j][k]).zfill(8)[-2:], 2) ^
sequence[index]) & 1)
                    index += 1

    # Convert the binary message to text
    message = ''.join([chr(int(binary_message[i:i+8], 2)) for i in range(0,
len(binary_message), 8)])
    return message
def generate_sequence(key, length):
    # Use logistic map equation for generating pseudo-random sequence

```

```

sequence = [key]
for i in range(length - 1):
    sequence.append(4 * key * (1 - key))
    key = sequence[-1]
return sequence

# Example usage
image_path = 'modified_image.png'
key = 0.5
message = decode(image_path, key)
print('Hidden message:', message)

```

Another Code Snippet:

```

import numpy as np
from scipy.special import lambertw

# Chaotic system - Logistic map
def logistic_map(x, r):
    return r*x*(1-x)

# Spreading function
def spread(x, seq):
    y = []
    for i in range(len(x)):
        y.append(x[i] + seq[i])
    return y

# Embedding function
def embed(img, y):
    # Embed y into blue channel of img
    b_channel = img[:, :, 0].reshape(-1)
    for i, v in enumerate(y):
        b_channel[i] = v
    return img

# Generate chaotic sequence
r = 3.9 # Chaotic system parameter
x = 0.5
seq = []

```

```

for i in range(10000):
    x = logistic_map(x, r)
    seq.append(lambertw(x).real)
# Spread and embed secret message
msg = [1, 0, 1, 0, 0, 1, 1, 0]
y = spread(msg, seq)
stego_img = embed(img, y)

```

Applications and usage

CSSIS has been used in various applications, such as digital watermarking, secure communication, and secret sharing. Additionally, CSSIS has been used to detect copyright infringement, detect malicious images, and protect data from unauthorized access.

1. **For secure communication**, CSSIS can be used to hide secret information in an image. This allows for secure communication between two parties, as the secret information is embedded in an image and can only be extracted with the correct key. The secret information is embedded in such a way that it is not visible to the human eye, but can still be detected.
2. **For secret sharing**, CSSIS can be used to split a secret into multiple parts and spread the parts across multiple images. This allows for the secret to be securely shared among multiple parties, as each party will only have access to one part of the secret. The secret information is embedded in such a way that it is not visible to the human eye, but can still be detected.
3. **For digital watermarking**, CSSIS can be used to embed a digital watermark into an image. This watermark is a unique identifier that can be used to identify the image and protect it from unauthorized access. The watermark can be embedded in the image in such a way that it is not visible to the human eye, but can still be detected.
4. **For copyright infringement detection**, CSSIS can be used to detect if an image has been tampered with. This can be used to identify copyright infringement and protect the original image. The algorithm can detect any changes that have been made to the image, such as cropping, resizing, or adding text.

The use of CSSIS for copyright infringement detection provides a means of detecting unauthorized copies or derivative works of an image, even if the copies have been altered or modified in some way. The digital watermark can be embedded in a way that is resistant to image processing techniques such as cropping, resizing, and compression, making it more difficult for infringers to remove or modify the watermark. However, it's worth noting that CSSIS is not foolproof and can be bypassed by skilled infringers who are able to extract the watermark or modify the image in such a way that the watermark becomes undetectable. Therefore, CSSIS should be used as a part of a larger copyright protection strategy that includes other measures such as legal action against infringers, monitoring of online content, and use of other digital rights management technologies.

5. **For malicious image detection**, CSSIS can be used to detect if an image contains malicious content. This can be used to protect users from malicious images, as the algorithm can detect any malicious content that has been embedded in the image.

Where it is used:

CSSIS has been used by a variety of organizations and companies, including Facebook, Twitter, and Microsoft. Additionally, it has been used by academic institutions such as the University of California, San Diego, the University of Texas at Austin, and the University of Toronto. It has been also used by government organizations such as the National Security Agency and the Department of Defense.

III. Functional Requirements

1. Messages Encoding

The system must allow users to encode messages into digital images using various steganography techniques, including LSB (Least Significant Bit) and DCT (Discrete Cosine Transform) techniques. The system should allow users to specify the input message and the image file to be used for encoding. The system should also allow users to choose the steganography technique to be used for encoding.

2. Messages Decoding

The system must allow users to decode messages from digital images using the same steganography technique that was used for encoding. The system should allow users to specify the image file from which to decode the message.

3. User Interface

The system must provide a user-friendly interface that allows users to interact with the system easily. The interface should include buttons and menus for encoding and decoding messages, selecting images, and choosing steganography techniques.

IV. Non-Functional Requirements

1. Performance

The system should be able to encode and decode messages from images quickly and efficiently. The system should be able to handle large image files and messages without slowing down or crashing.

2. Security

The system should provide a high level of security for encoded messages. The system should ensure that the encoded messages are not easily detectable and that they cannot be easily deciphered by unauthorized users.

V. Objectives of the Proposed Solution:

The primary objectives of the image steganography system are as follows:

- Develop an intuitive user interface for interacting with the system.
- Implement steganographic algorithms to embed and extract secret messages within digital images.
- Ensure minimal distortion to the cover image to maintain visual quality.
- Enhance the system's security by incorporating encryption techniques for the secret message.
- Evaluate the system's performance in terms of capacity, robustness, and security.

VI. Design of the components

1. Description:

- **Encryption Component:**

The writer will first enter their text into the encryption component. The encryption component will encrypt the text using a secure encryption algorithm, such as AES encryption. The output of the encryption component will be the encrypted text.

- **Image Component:**

The writer will select an image that they want to hide the encrypted text in. The image component will take in the selected image and the encrypted text and then hide the encrypted text inside the image using the LSB algorithm.

- **Decryption Component:**

The user will enter the encoded image into the decryption component. The decryption component will then extract the hidden encrypted text from the image using the LSB algorithm. The output of the decryption component will be the encrypted text.

- **Decryption Key:**

The user will provide the decryption key to the decryption component to decrypt the encrypted text.

- **Decryption Algorithm:**

The decryption component will use a secure decryption algorithm, AES decryption, to decrypt the encrypted text.

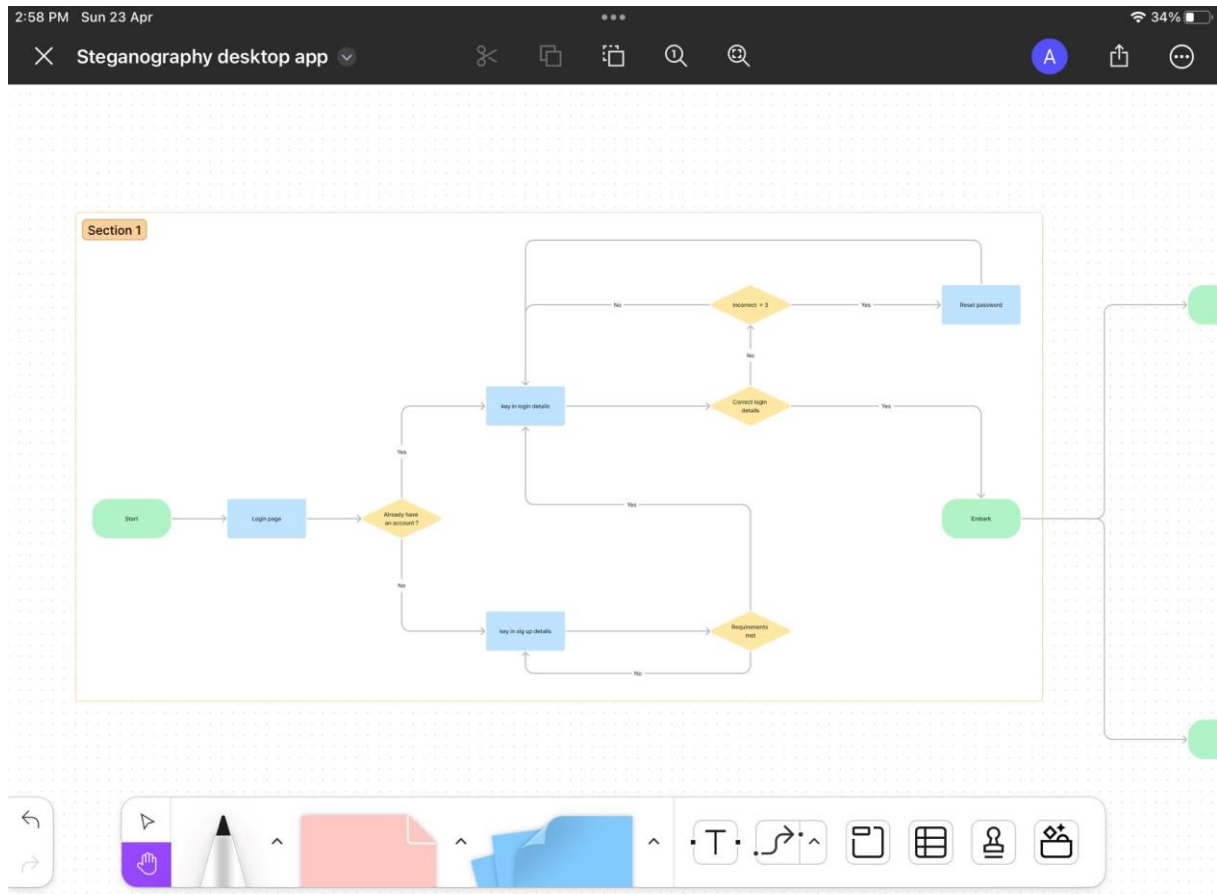
- **Display Result Component:**

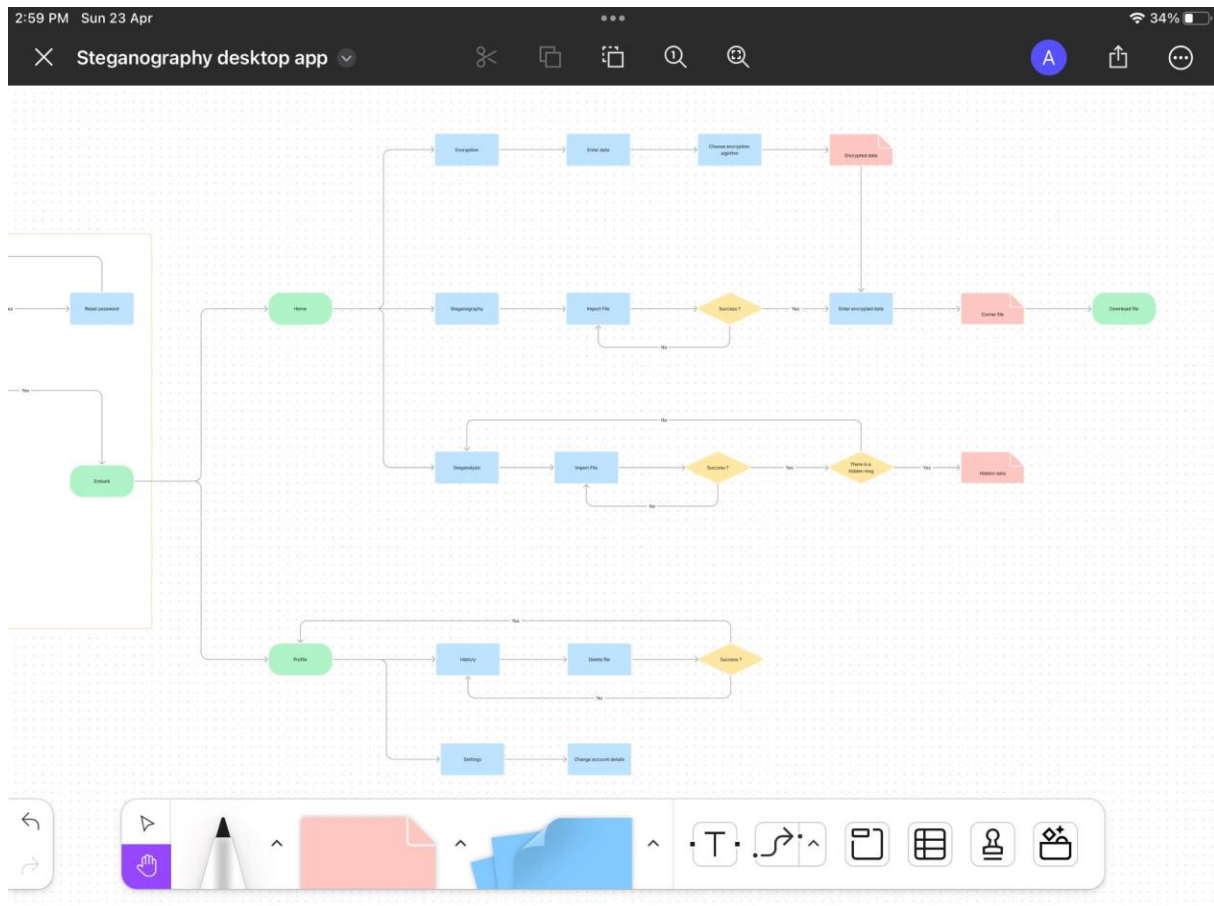
The result of the decryption component, which will be the decrypted text, will be displayed to the user in the display result component.

2. User Flow:

The writer (**user1**) enters the text into the encryption component. The encryption component encrypts the text and outputs the encrypted text. The writer selects an image and inputs it along with the encrypted text into the image component. The image component hides the encrypted text inside the image using the LSB algorithm and outputs the encoded image. Now The receiver(**user2**) inputs the encoded image into the decryption component along with the decryption key.

The decryption component extracts the hidden encrypted text from the image using the LSB algorithm and decrypts it using the decryption algorithm. The result of the decryption component, which is the decrypted text, is displayed to the user in the display result component.





3. Use Case:

- **Actors:**

Writer (User1): This actor inputs the text to be encrypted, selects an image to encode the text, and saves the encoded image.

Receiver (User2): This actor inputs the encoded image to be decoded and inputs the decryption key to decrypt the hidden message.

- **Use Cases:**

Encrypt Text: This use case describes the process of encrypting the input text using a secure encryption algorithm.

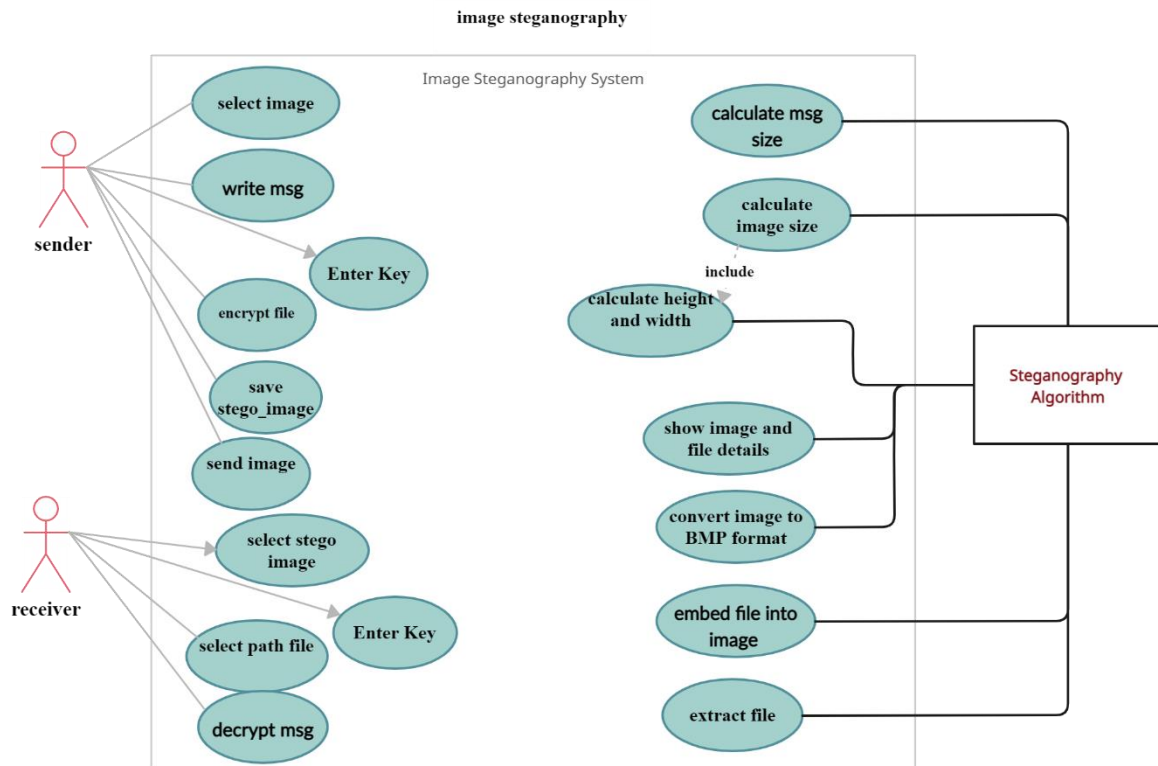
Select Image: This use case describes the process of selecting an image to hide the encrypted text in.

Encode Image: This use case describes the process of hiding the encrypted text inside the image using the LSB algorithm.

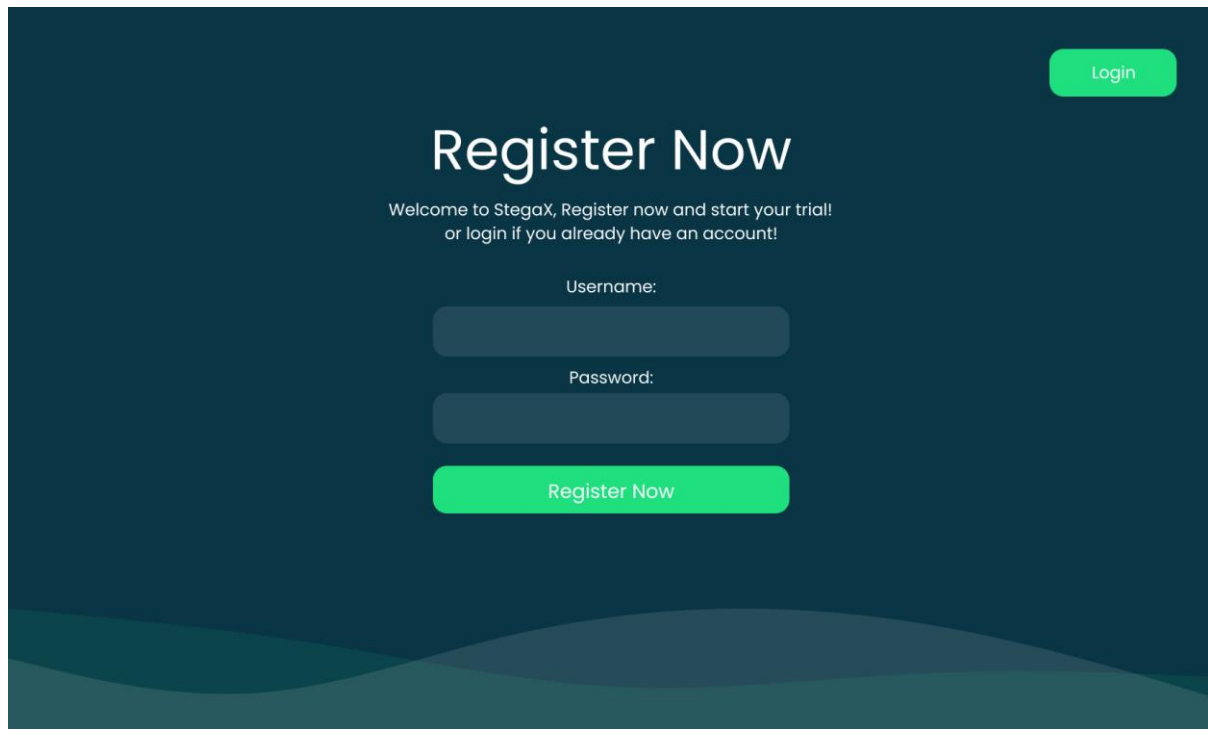
Save Encoded Image: This use case describes the process of saving the encoded image.

Decode Image: This use case describes the process of extracting the hidden encrypted text from the encoded image using the LSB algorithm.

Decrypt Text: This use case describes the process of decrypting the encrypted text using a secure decryption algorithm.



4. Fast Prototype (UI):



This is a UI mockup for a registration page. The background is a dark teal color with a subtle, abstract landscape pattern at the bottom. In the top right corner, there is a small, rounded green button with the text "Login". The main heading "Register Now" is centered in a large, white, sans-serif font. Below the heading, a line of smaller white text reads: "Welcome to StegaX, Register now and start your trial! or login if you already have an account!". The form consists of two stacked input fields with light gray borders and rounded corners. The first field is preceded by the label "Username:" and the second by "Password:". Below these fields is a prominent, rounded green button with the text "Register Now" in white.

Login

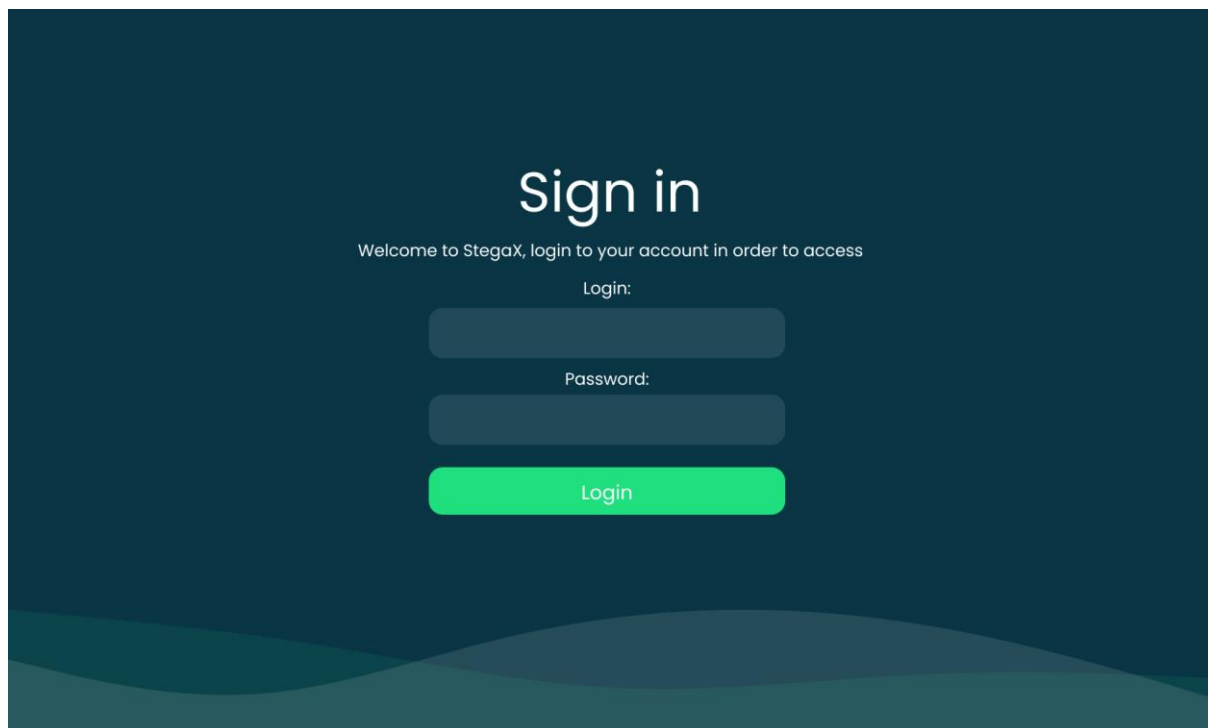
Register Now

Welcome to StegaX, Register now and start your trial!
or login if you already have an account!

Username:

Password:

Register Now



This is a UI mockup for a sign-in page. It features the same dark teal background and abstract landscape pattern at the bottom as the registration page. The main heading "Sign in" is centered in a large, white, sans-serif font. Below the heading, a line of smaller white text reads: "Welcome to StegaX, login to your account in order to access". The form consists of two stacked input fields with light gray borders and rounded corners. The first field is preceded by the label "Login:" and the second by "Password:". Below these fields is a prominent, rounded green button with the text "Login" in white.

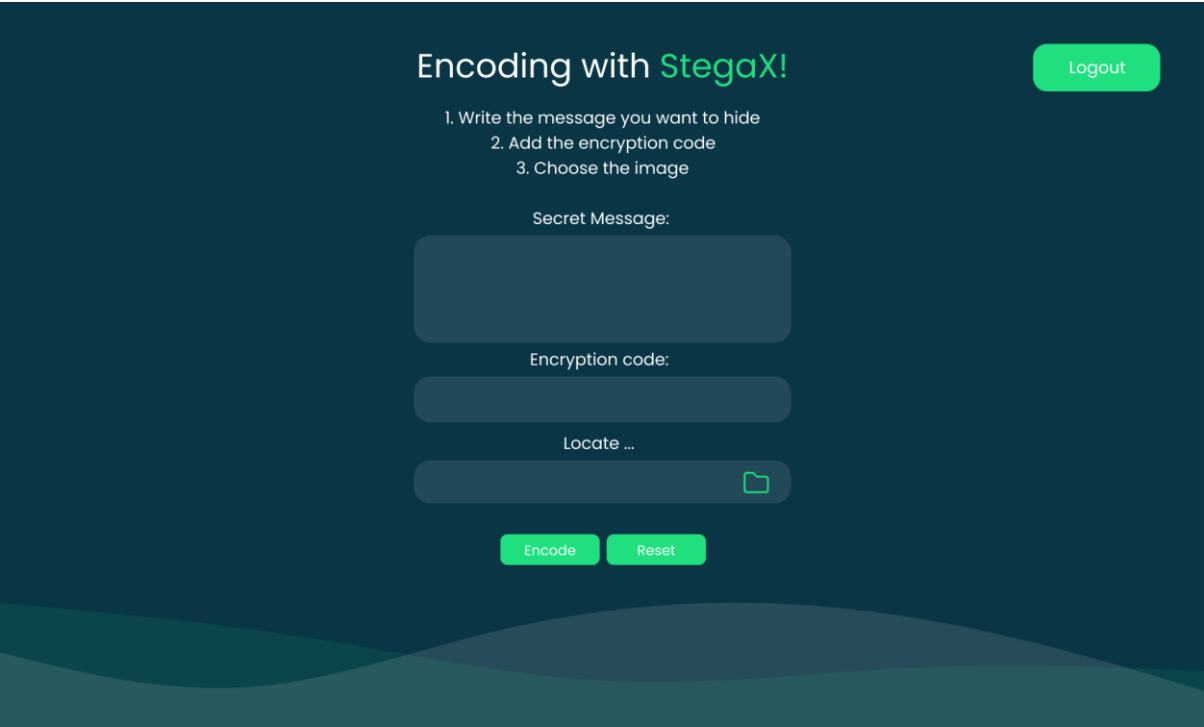
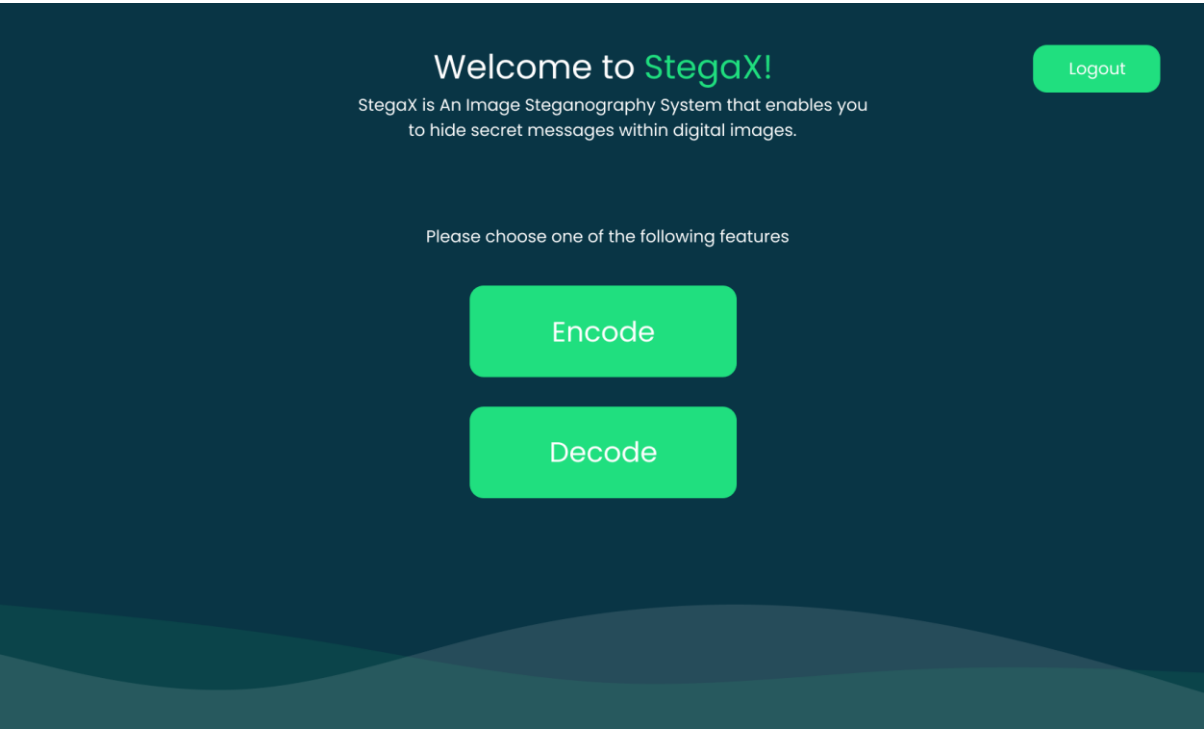
Sign in

Welcome to StegaX, login to your account in order to access

Login:

Password:

Login



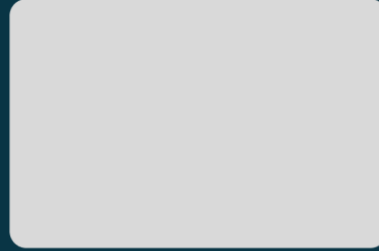
Encoding with StegaX!

Logout

Image Successfully encoded!



Original Image



Encrypted Image

Save

Decode

Home

VII. Methodology

The image steganography system is built using the following steps:

1. Image Preprocessing

The system initially preprocesses the cover image to ensure compatibility with the steganographic techniques. This involves converting the image to a suitable format, such as PNG or BMP, and performing any necessary resizing or normalization.

2. Secret Message Encryption

To enhance security, the system encrypts the secret message using a symmetric or asymmetric encryption algorithm. This step ensures that even if the hidden information is discovered, it remains unintelligible without the decryption key.

3. Embedding the Secret Message

The system uses a steganographic algorithm, such as Least Significant Bit (LSB) substitution or the Spread Spectrum Technique (SST), to embed the encrypted secret message within the cover image. The algorithm selectively modifies the pixel values of the image to conceal the message while minimizing visual distortion.

4. Extraction of the Secret Message

To retrieve the hidden message, the system applies the reverse process by utilizing the same steganographic algorithm. The pixels of the modified image are analyzed to extract the embedded message, which is then decrypted using the appropriate decryption key.

VIII. Project Tools:

1. Cryptography: Message Encryption and Decryption:

This task will be performed using one, or a combination, of these libraries, depending on the solution we will judge to be the best.

- **Cryptography:** Cryptography is a popular Python library that provides support for various cryptographic algorithms, such as AES, RSA, and HMAC. It can be used to encrypt and decrypt messages and files.
- **PyCrypto:** PyCrypto is another Python library that provides support for various cryptographic algorithms, such as AES, RSA, and SHA. It can be used to encrypt and decrypt messages and files.
- **PySeCrypt:** PySeCrypt is a Python library that provides an easy-to-use interface for encrypting and decrypting messages using various cryptographic algorithms, such as AES and Blowfish.

2. Image Steganography:

Different Python libraries can be used, namely:

- **Pillow:** Pillow is a Python library that provides an easy-to-use interface for opening, manipulating, and saving various image file formats. It can be used to implement various steganographic techniques, such as LSB, DCT, and DWT.
- **NumPy:** NumPy is a popular Python library for scientific computing. It provides support for arrays and matrices, which can be used for image processing and manipulation. NumPy can be used to implement various steganographic techniques, such as DCT and DWT.
- **OpenCV-Python:** OpenCV-Python is a Python binding of the OpenCV library, which is a popular open-source computer vision library. It provides support for various image processing functions, including image filtering, edge detection, and object detection. OpenCV-Python can be used to implement steganography techniques that involve manipulating image pixels.

- Stegano: Stegano is a Python library that provides support for steganography. It includes support for various steganographic techniques, such as LSB, DCT, and DWT. Stegano provides an easy-to-use interface for hiding and extracting data from image files.
- StegExpose: StegExpose is a tool that can be used to detect steganography in image files. It provides support for detecting various steganographic techniques, including LSB, DCT, and DWT.

3. Creating a User Interface:

- Tkinter: Tkinter is a built-in Python library that provides support for creating GUI applications. It provides various widgets such as buttons, labels, and text boxes, which can be used to create a user interface for your steganography project.
- Subprocess: We used this library to execute a system command or launch an external program from within a Python script, open other interfaces when clicking on the button.
- Pathlib: The pathlib module provides an object-oriented approach to handle file system paths. The Path class is the main class in the pathlib module and allows you to perform various operations on file paths, such as joining paths, checking file existence, manipulating paths, and more.

4. Managing the database:

- SQLite3: it is used to manage SQLite databases. This module provides an interface to interact with SQLite databases and perform various database operations. Note that we will use this database for the registration and login!

We chose to ensure a double protection for our data, by performing both the cryptography and steganography tasks. We know that even though the steganography algorithms may seem quite powerful, they can present some weaknesses. The strategy of encrypting the hidden message aims to avoid any risk of interception.

IX. Project Phases:

Project Initiation Phase (All the Team)

Task 1: Project Kickoff

Task 2: Research and analysis

- Subtask 2.1: Identify Steganography algorithms
- Subtask 2.2: Conduct a deep research about the usage of the system
- Subtask 2.3: Document the project requirements

Design Phase

Task 3: System Architecture Design

- Subtask 3.1: Identify the components of the system
- Subtask 3.2: Define the interaction between components
- Subtask 3.3: Design the data flow and storage mechanisms

Task 4: User Interface Design

- Subtask 4.1: Create wireframes/mockups for the user interface
- Subtask 4.2: Incorporate user feedback and refine the designs
- Subtask 4.3: Finalize the user interface designs

Development Phase

Task 5: Backend Development

- Subtask 5.1: Implement image encryption algorithm
- Subtask 5.2: Implement steganography algorithm
- Subtask 5.3: Integrate encryption and steganography modules
- Subtask 5.4: Link the database to the system

Task 6: Frontend Development

- Subtask 6.1: Develop login and registration functionality
- Subtask 6.2: Implement image selection and encoding interface
- Subtask 6.3: Build image decoding interface
- Subtask 6.4: Implement decoding Interface

Task 7: Testing and Bug Fixing

- Subtask 7.1: Develop test cases for each module
- Subtask 7.2: Identify and fix bugs and issues

Deployment Phase

- Task 8: Deployment Planning
 - Subtask 8.1: Define deployment environment and requirements
 - Subtask 8.2: Prepare deployment checklist
- Task 9: System Deployment
- Task 10: User Documentation
 - Subtask 10.1: Create user documentation and guides

Project Closure Phase

- Task 11: Finalize Project Documentation

H14

fx

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W					
1	Steganography Project Schedule																											
2																												
3																												
4	Start Date: 5/15/2023 (Monday)													Display Week: 1					Week 1					Week 2				
5														15 May 2023					22 May 2023									
6														15	16	17	18	19	20	21	22	23	24	25	26	27	28	
7	WBS	Task	Lead	Start	End	Days	% Done	Work Days	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su						
8	1	Cryptography Design	Imen	Mon 5/15/23	Wed 5/17/23	3	0%	3																				
9	2	Cryptography Implementation	Imen	Thu 5/18/23	Fri 5/19/23	2	0%	2																				
10	3	Steganography Design	Aziz	Mon 5/15/23	Sat 5/20/23	6	0%	5																				
11	4	Link the database to the system	Takwa	Sun 5/21/23	Sat 5/27/23	7	0%	5																				
12	13	User Interfaces Design	Rayen	Mon 5/15/23	Wed 5/17/23	3	100%	3																				
13																												
14																												

X. Implementation Details:

The image steganography system is implemented using the following technologies:

1. Programming Language:

The system is developed using a programming language such as Python, which offers a wide range of libraries and tools for image processing and cryptography.

2. User Interface:

The system incorporates a user-friendly graphical interface developed using a framework like Tkinter or PyQt. The interface allows users to select cover images, enter secret messages, and perform embedding and extraction operations.

3. Image Processing Libraries:

To manipulate and process images, the system utilizes image processing libraries such as OpenCV or PIL (Python Imaging Library). These libraries provide functions for image loading, pixel manipulation, and format conversion.

4. Encryption Libraries:

For message encryption, cryptography libraries like PyCrypto or cryptography.io are used. These libraries offer various encryption algorithms such as AES or RSA for securing the secret message.

XI. Evaluation:

The image steganography system is evaluated based on the following criteria:

1. Capacity:

The system's capacity refers to the maximum size of the secret message that can be embedded within an image. The evaluation measures the system's ability to handle messages of varying lengths without significantly degrading the image quality. Several test cases with different message lengths are used to assess the system's capacity and ensure it meets the desired requirements.

2. Robustness:

The robustness of the system is evaluated by subjecting the steganographic algorithm to various image modifications and attacks. These include resizing, cropping, compression, and common steganalysis techniques. The system's ability to maintain the integrity of the hidden message and resist detection is examined. Robustness is an essential aspect of ensuring the system's effectiveness and security.

3. Security:

The security of the system is evaluated by assessing its resistance against known steganalysis methods and attacks. Various steganalysis techniques, such as statistical analysis, histogram analysis, and visual inspection, are applied to detect the presence of hidden information. The system's ability to withstand these attacks and maintain the confidentiality of the embedded message is examined. Additionally, the encryption strength of the secret message is assessed to ensure the message remains secure even if the image is compromised.

XII. Conclusion:

The image steganography system successfully achieves the objectives outlined in this report. It provides a user-friendly interface for embedding and extracting text messages within digital images. The system demonstrates good capacity, robustness against image modifications and attacks, and sufficient security through message encryption.

The system's implementation, evaluation, and future possibilities showcase its potential as a valuable tool for confidential communication in various domains, including data security, digital forensics, and covert operations.

XIII. Future Work:

Although the image steganography system has met the project's objectives, there are opportunities for future enhancements and extensions. Some potential areas for further research and development include:

- Integration of more advanced steganographic techniques to improve capacity and robustness.
- Implementation of additional encryption algorithms to offer a broader range of security options.
- Exploration of multi-image steganography techniques for embedding larger messages or multiple messages within a set of images.
- Development of a mobile application or web-based platform to increase accessibility and usability.

XIV. References

<https://www.edureka.co/blog/steganography-tutorial>

<http://www.diag.uniroma1.it/~bloisi/steganography/isc.pdf>

<https://data-flair.training/blogs/python-image-steganography-project/>

<https://projectgurukul.org/python-image-steganography/>

<https://www.geeksforgeeks.org/image-based-steganography-using-python/>

<https://betterprogramming.pub/image-steganography-using-python-2250896e48b9>

<https://ieeexplore.ieee.org/document/9335027>