Stegame

Introduction:

We concluded our previous deliverable by comparing various image steganography methods, as outlined in the table below.

	LSB	Transform Domain	Spread Spectrum	Statistical Techniques	Distortion Techniques	File and Pallet Embedding
Imperceptibility	High*	High	High	Medium*	Low	High*
Robustness	Low	High	Medium	Low	Low	Low
Payload Capacity	High	Low	High	Low*	Low	High

Our analysis highlighted the trade-offs associated with LSB techniques: while they excel in concealing large amounts of information (high payload capacity), they often fail to adequately camouflage the statistical properties of the image. This vulnerability makes them susceptible to statistical attacks and image manipulation, compromising their robustness.

To leverage the simplicity and high payload capacity of the LSB algorithm without compromising on robustness, we propose the development of a user-friendly web-based application. This application will enable users to embed secret text messages using an enhanced version of the LSB algorithm. This enhanced algorithm will retain the simplicity of implementation while offering a higher level of security against attacks.

The purpose of Stegame is to provide users with a straightforward and efficient means of embedding secret messages within images, addressing the challenge of balancing simplicity with security in image steganography.

System Architecture:

Components:

Stegame's architecture consists of several key components that work together to enable users to embed secret text messages within images securely. The components include:

1. User Interface (UI): The UI provides an intuitive and user-friendly interface for users to interact with the system. It allows users to upload images, enter secret messages, and perform embedding and extraction operations.

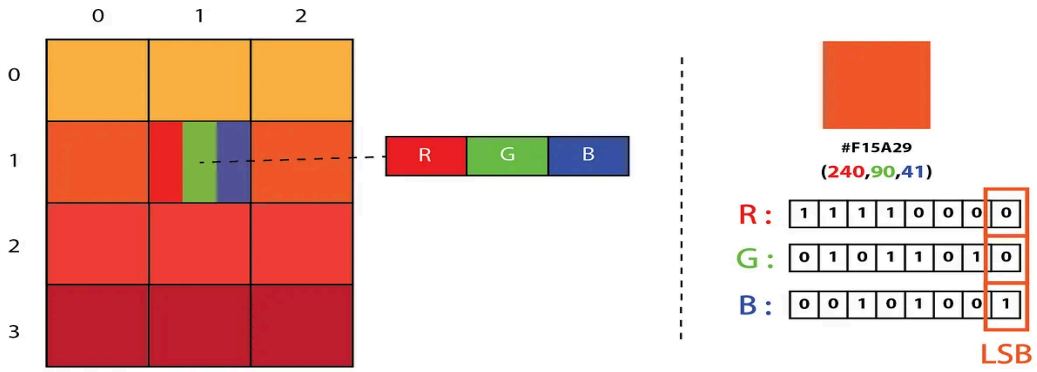
2. Encoding/Decoding Module: This module is responsible for encoding data into images and decoding hidden data from images. It uses an enhanced version of the LSB algorithm to embed and extract messages, ensuring high payload capacity and security.

Terminology

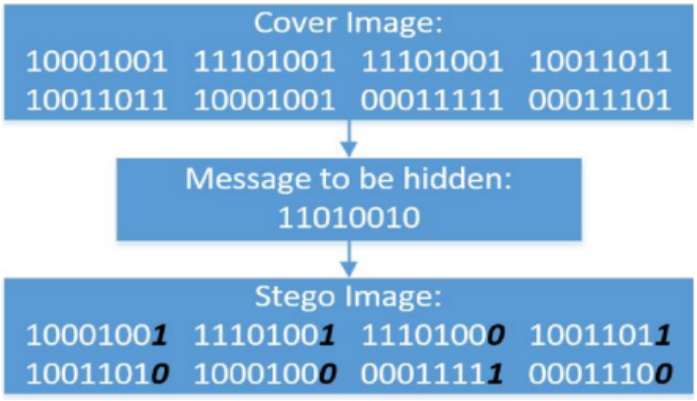
- *LSB Technique*

The least significant bit embedding method is a widely used and simple method to apply. In this method, the bits of the message to be hidden are placed one by one to the least significant bit of each byte of pixels forming a cover image.

To understand better, let's consider a digital image to be a 2D array of pixels. Each pixel contains values depending on its type and depth. We will consider the most widely used modes — **RGB(3x8-bit pixels, true-color)** and **RGBA(4x8-bit pixels, true-color with transparency mask).** These values range from 0–255, (8-bit values).



We can convert the message into decimal values and then into binary, by using the ASCII Table. Then, we iterate over the pixel values one by one, after converting them to binary, we replace each least significant bit with that message bits in a sequence.
To decode an encoded image, we simply reverse the process. Collect and store the last bits of each pixel then split them into groups of 8 and convert it back to ASCII characters to get the hidden message.



An example of LSB steganography technique

3. Encryption Module: The encryption module encrypts the secret messages before embedding them into images. This adds an extra layer of security, ensuring that even if the hidden messages are discovered, they cannot be read without the decryption key.

Terminology

- Logistic Map

Chaos-based encryption programs basically generate chaotic equations and generate a long random number sequence such as pseudo-random number generators.One of the simplest and most studied nonlinear systems is the logistic map. The logistic map is a mathematical function that is often used in chaos theory and dynamical systems. In the context of encryption, the logistic map can be used to generate a sequence of pseudo-random numbers that can be used as a key to encrypt and decrypt messages.

Here's a simple explanation of how the logistic map works and how it can be used in text encryption:

1. Logistic Map Function:
The logistic map function is defined by the formula: $X(n + 1) = r{\cdot}Xn{\cdot}(1 - Xn)$, where Xn is the current value of the variable X, $X(n + 1)$ is the next value of X, and r is a parameter that determines the behavior of the function.

2. Generating Pseudo-Random Numbers:
By iterating the logistic map function with different initial values of X and values of r, you can generate a sequence of pseudo-random numbers. These numbers can be used as a key to encrypt and decrypt messages.

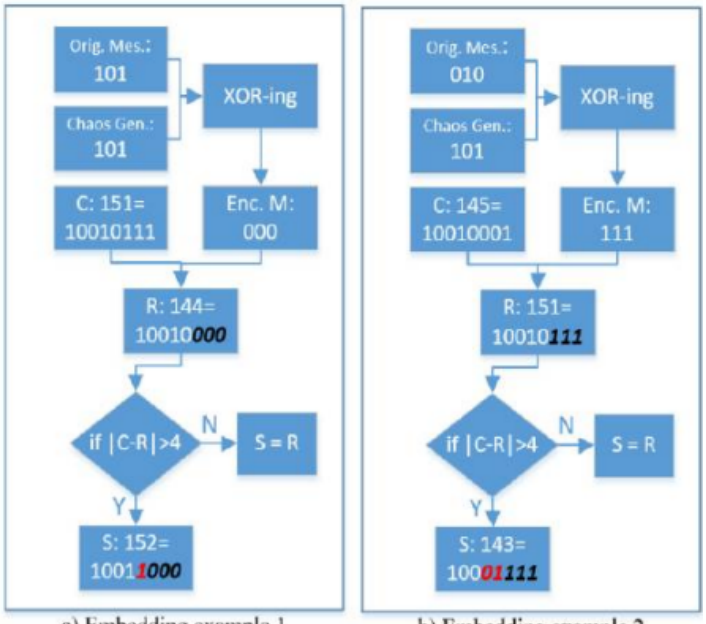
3. Encrypting Messages:
To encrypt a message, you can convert each character in the message to its ASCII value and XOR it with a corresponding pseudo-random number generated by the logistic map. This process masks the original message, making it difficult for unauthorized users to read.

4. Decrypting Messages:
To decrypt the encrypted message, you can XOR each masked ASCII value with the same pseudo-random number to recover the original message.

Overall, the logistic map provides a simple yet effective way to generate pseudo-random numbers that can be used for text encryption. Its chaotic nature ensures that the sequence of numbers appears random, enhancing the security of the encryption process and thus complexity or robustness of stego-image against attacks.

Working Flows:

1. Embedding Workflow:
   Embedding Algorithm (n-LSB on 24 bit RGB image)
   - Using the logistic map, random numbers are produced up to the number of message bits to be embedded. The initial conditions x and u are used as keys in the communication.
   - Since the chaos generator produces numbers between 0 and 1, these numbers are first multiplied by 10 until the integer is reached.
   - Then, the least significant bits of these integers are taken and used as random numbers.
   - The encrypted message bits are obtained by XOR-ing the random numbers with the original message bits.
   - The least significant n bits of each pixel's R, G and B channels are sequentially replaced with the bits of the encrypted message.
   - If the difference between the newly created R, G and B values and the original R, G and B values is more than 2n/2, then this difference is reduced by; Checking and modifying all consecutive bit starting from (n+1)th bit till the 8th bit (most significant bit) unless it has value of 1. When the bit with value of 1 is met then change its value to 0 and stop the operation. If we have all 0s starting from (n+1)th bit to 8th bit then don't do any change on these consecutive bits.
   - If the difference between the newly created R, G and B values and the original R, G and B values is less than -2n/2, then this difference is increased by; Checking and modifying all consecutive bit starting from (n+1)th bit till the 8th bit (most significant bit) unless it has value of 0. When the bit with value of 0 is met then change its value to 1 and stop the operation. If we have all 1s starting from (n+1)th bit to 8th bit then don't do any change on these consecutive bits.



a) Embedding example 1 b) Embedding example 2

Let's check how the proposed method works on 3-LSB:
Assume that we have 111 (3 bits) encrypted data to embed into the R channel whose last 3 bits is 000. For this case the change in cover image will be 7 (bigger than 2 exponent 3/2). This will affect the visibility or sense of a human on the image. The proposed method will check the 4th bit of cover image and if it is 1 then it will change it to 0 and then stop the operation. Changing 4th bit value from 1 to 0 means reducing the value of related pixel -8. Total change of related pixels will be 7-8=-1 instead of 7, which means to increase PSNR ratio of stego-image.

PSNR ratio:One of the most well-known and widely used measures of comparing two images is the Peak Signal to Noise Ratio (PSNR). PSNR is an engineering term for the proportion of the maximum possible power of original image to the power of the differences between original image and stego image. The unit of PSNR is decibel (dB). A higher PSNR value means that the stego image is less distorted.

2. Extraction Workflow:
   Extraction Algorithm
   - The least significant 3 bits of each pixel of R, G and B channels of the stego image are fetched in order.
   - Random numbers are generated using a chaotic random number generator and initial values x and u. Since the chaos generator produces numbers between 0 and 1, these numbers are first multiplied by 10 until the integer is reached.
   - Then, the least significant bits of these integers are taken and used as random numbers.
   - The original message bits are obtained by XORing the random numbers with the extracted message bits.

Roles/Users:

1. Non-technical Users: These users may include individuals who wish to send secret messages securely without requiring technical expertise. They may include:
- Students or researchers who want to hide messages in images for academic or personal use.
- General users who want to share confidential information securely, such as passwords or personal messages.
2. Technical Users: These users may have a basic understanding of steganography and encryption techniques and may include:
- Security enthusiasts or hobbyists who are interested in experimenting with steganography.
- Professionals who require a simple and efficient tool for secure communication, such as journalists, activists, or business professionals.

Data/Message Exchange:

- User uploads an image and enters a secret message for embedding.
- User uploads an image with a hidden message for extraction.

Conclusion:

The proposed solution addresses the challenge of balancing simplicity with security in image steganography by providing a user-friendly web-based application that enables users to embed secret text messages using a combination of an enhanced LSB algorithm insertion and one dimensional logistic based (chaos theory) encryption.This solution offers a straightforward and efficient means of embedding secret messages within images while maintaining a high level of security against attacks.