PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

UE23EC342AC1 - DIP

Aug - Dec 2025

Report on

Hidden Communication (Steganography and Integrity Check)

Submitted by

Siddharth Bhat (PES1UG23EC296)

Shreyas B (PES1UG24EC812)

Sauradipta Basu(PES1UG23EC280)

Course Instructor: Dr. Shikha Tripathi

Department of Electronics and Communication Engineering

Table of Contents: -

INTRODUCTION:

The objective of the project is to demonstrate and illustrate the art of hidden communication using steganography.

The project aims to hide text or image given as input by the user in a selected carrier image of the user's choice. The project also gives provision to perform hidden communication via steganography, with the help of a password & XOR encryption.

The report primarily contains 3 methods of steganography as follow:

1. Text encryption in image with LSB manipulation.
    a. Grayscale Image
    b. Colour Image
2. Encryption of an image into a carrier image incorporating a XOR based password/key along with LSB manipulation.

1(a) Text encryption in carrier image with LSB manipulation

THEORY:

1. The objective of steganography is to hide any form of secret data (text in this case) in a selected carrier of any form of digital media (image in this case)

2. LSB (Least Significant Bit) manipulation exploits the fact that a small change in the last bit of a pixel value of an image, will not lead to large significant or discernable change in the carrier image.

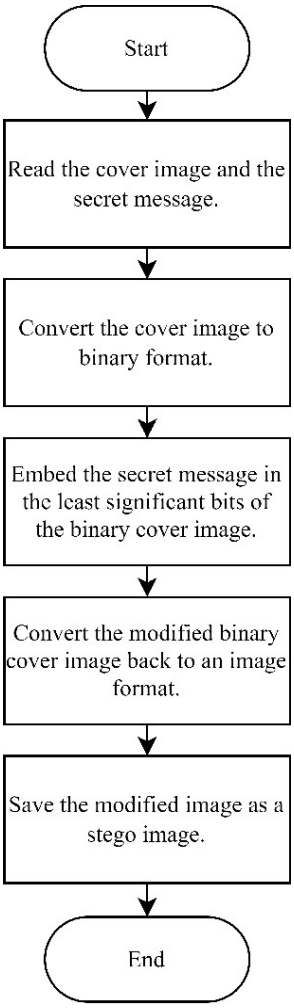
3. For example: - If the LSB is 0 (black) then changing it to 1 will not cause any drastic changes to the colour of the pixel. The pixel will only become very slightly gray.

ALGORITHM:

1. Input carrier image is read and converted from RGB to Grayscale.
2. Input message is taken from the user and converted to their ascii values using uint8. The ascii values are then converted to binary form using dec2bin.
3. The array of binary values is reshaped to one column such that it forms a long 1D array. The number of characters are then compared with the size of the image to ensure encryption.
4. For the process of encryption, the carrier image undergoes bitand with 254 such that only all LSBs are cleared. The binary encoded message bits are added to the corresponding carrier image.
5. The image with hidden text is saved onto a desired location.
6. For the decryption the length of the secret message string is taken as input.
7. The length is multiplied by 8 as each character takes 8 bits.
8. Floor division is done to extract the LSBs and stored in a sequence.Since each char is assigned 8 bits, a weight array is taken which contains values from 2^0 to 2^(8-1).

9. The bit sequence is arranged to its original length and each of the bits are multiplied with the weight array and the obtained ascii values are converted to char.
10. This provides the original message which is printed in the terminal.

FLOWCHART:



CODE:

ENCRYPTION CODE:

```
img = imread('C:\Users\saura\Downloads\DIP\Images-DIP\Mona.jpg');
[x,y,z] = size(img);
if z > 1
   img = rgb2gray(img);
end
hidimg = img;
msg = input('Please enter the message you want to hide: ','s');
asc_val = uint8(msg);
asc_to_bin = dec2bin(asc_val,8);        % convert obtained ascii values to bits
bin_seq = reshape(asc_to_bin.', 1, []);  % reshape all bits into single 1-D array
len = length(bin_seq);
if length(msg) > (x*y)
   disp('Chosen image is not large enough to hide the message.');
end
count = 1;
for i = 1:x
   for j = 1:y
      if count <= len
         bitmsg = str2double(bin_seq(count));
         hidimg(i,j) = bitand(hidimg(i,j), 254);   % clear LSB
         hidimg(i,j) = hidimg(i,j) + bitmsg;       % embed bit onto LSB
         count = count + 1;
      end
   end
end
subplot(1,2,1)
imshow(img);
title('Original Image');
subplot(1,2,2)
imshow(hidimg);
title('Encrypted Image');
```

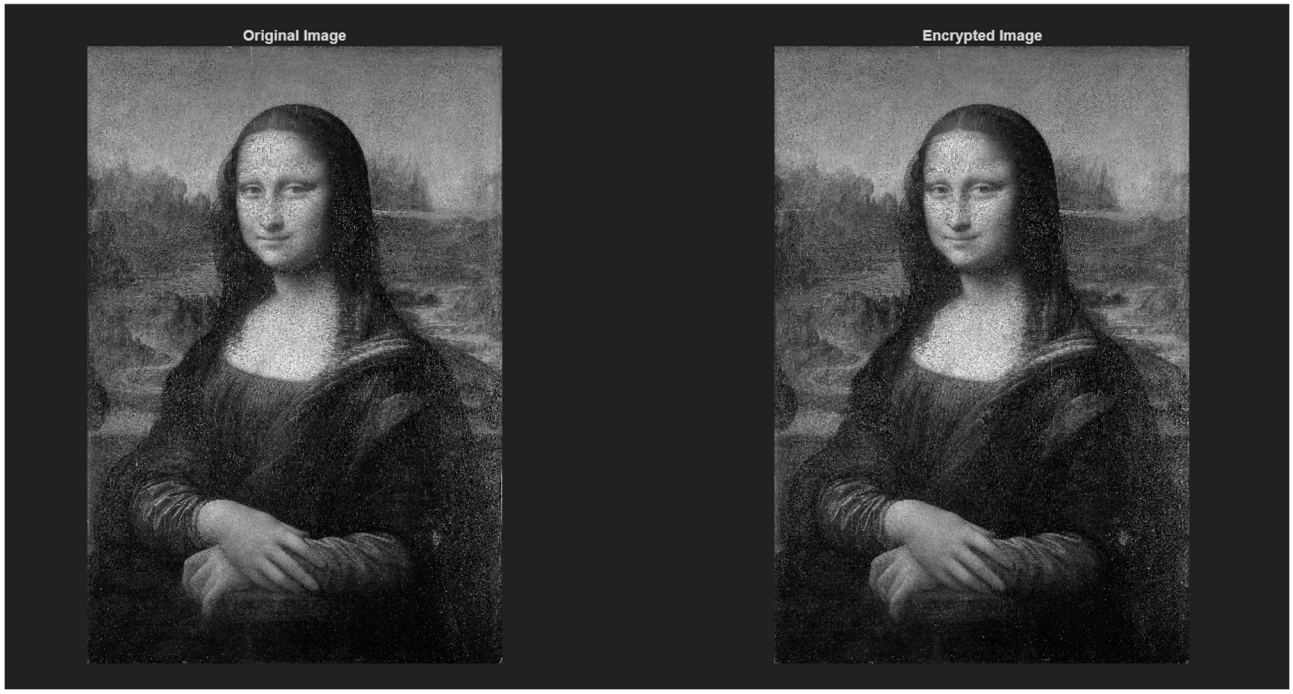
imwrite(hidimg,'C:\Users\saura\OneDrive\ドキュメント
\MATLAB\DIP\output.png');

DECRYPTION CODE:

```
clc;
clear;
len=str2double(input('Enter the length (character count) of the message you are
looking for: ','s'));
len=len*8;
hidimg=imread("C:\Users\saura\OneDrive\ドキュメン\MATLAB\DIP\output.png");
[x,y,z] = size(hidimg);
if z > 1
  hidimg = rgb2gray(hidimg);
end
count=1;
bitseq='';
for i=1:x
  for j=1:y
    if count<=len
      %Retrieve the LSB of the pixel value
      LSB=mod(hidimg(i,j),2);
      bitseq(count,1) = LSB;
      count=count+1;
    end
  end
end
%Converting the bit sequence into the original message using weighted
%values
weights = [ 128 64 32 16 8 4 2 1 ];
% elements are taken column wise from bitseq, each char has 8 bits so 8 rows
message_matrix = reshape(bitseq,8,len/8);
original_message = char(weights*message_matrix);
```

disp(['The original message is: ', original_message]);

OUTPUT:




```
>> steg_im_txt
Please enter the message you want to hide: Mona Lisa's Secret
```

```
Enter the length (character count) of the message you are looking for: 18
The original message is: Mona Lisa's Secret
```

OBSERVATIONS:

1. The text is successfully hidden in the carrier image without altering the image in a noticeable way.
2. The text can easily be extracted from the carrier image just by knowing the length of the hidden/encrypted text/string.
3. The image output.png cannot be differentiated easily from the input grayscale image.

CONCLUSIONS:

1. The desired text can be hidden in the desired carrier image very easily.

2. Not much noticeable change observed in carrier image.

3. Easy technique to hide text and other media in desired digital media such as images.

1(b) LSB-Based Text Steganography in Colour Images

Text-in-image steganography involves concealing a secret text message inside a colour image, known as the cover image, by making minor modifications to the pixel values. A colour image is composed of three channels—Red, Green, and Blue (RGB)—each containing 8-bit intensity values. Because the least significant bit (LSB) of each channel contributes very little to the overall colour, changing it causes almost no visible difference in the final image.

In the LSB method, the bits of the secret text are sequentially embedded into the LSBs of the pixel values across one or more RGB channels. Since only the last bit of each colour component is altered, the visual appearance of the cover image remains nearly identical to the original. This makes LSB substitution a simple yet highly effective technique for data hiding.

During extraction, the embedded LSBs are read back in the same order to reconstruct the binary representation of the hidden text. Because the method modifies only minimal pixel information, the recovered text remains accurate, while the carrier image retains high visual quality.

ALGORITHM

Embedding Phase:

1. Read the colour carrier image.
2. Convert the secret text into its ASCII values.
3. Convert each ASCII value into an 8-bit binary sequence.
4. Flatten the pixel values of the carrier image into a sequence of R, G, and B channel values.

5. For every bit of the secret message:
   A. Take the next carrier pixel value.
   B. Clear its LSB (pixel & 254).

C. Take the next bit from the secret text (0 or 1).
D. Add this bit to the cleared pixel value to form the stego pixel.

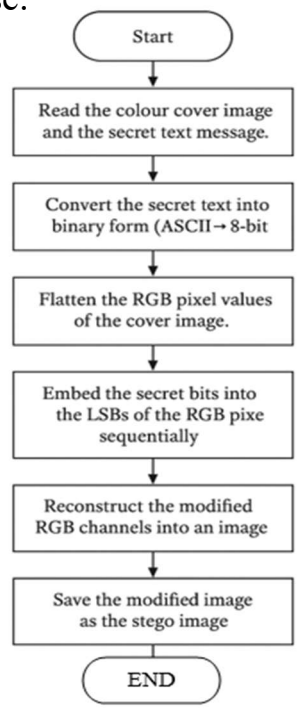
6. Replace the modified pixel values back into the R, G, B channels to form the stego image.
7. Save the stego image.

Extraction Phase:

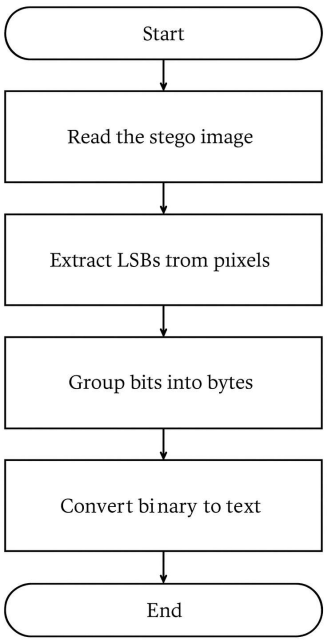
1. Read the stego image and separate it into R, G, B channels.
2. Flatten all pixel values into a single sequence.
3. For every pixel used in embedding:
   a) Extract the LSB (pixel & 1).
   b) Store this bit in sequence.
4. Group every 8 bits to form one ASCII value.
5. Convert the ASCII values back into characters to recover the secret text.

FLOWCHART:

Embedding Phase:



Extraction Phase:



CODE:

Embed Code:

```
cover_filename = 'C:\Users\Shreyasbavimj\Downloads\yogi reggie.jpg';
output_filename = 'stego_image.png';
message = 'frame by frame'; % secret message
numLSB = 1; % number of LSBs to use per channel pixel (1 is safest)
%
cover = imread(cover_filename);
[H, W, C] = size(cover); % C = 1 for grayscale, 3 for RGB
% Flatten pixel data channel-wise into a vector of unsigned integers
pixel_count = H * W * C;
pix = cover(:);% column vector (uint8)
% Prepare message bits with a header length
msg_bytes = uint8(message); % ASCII bytes
msg_len = numel(msg_bytes); % number of bytes
% We'll store length as a 32-bit unsigned integer header (allows up to ~4GB
message)
len_header = typecast(uint32(msg_len), 'uint8'); % 4 bytes representing length
payload = [len_header(:); msg_bytes(:)];% first 4 bytes = length, then message
bytes
% Convert payload to bit stream (MSB first for each byte)
bits = reshape(dec2bin(payload,8).' - '0', [], 1); % column vector of 0/1, length
= 8*(4+msg_len)
% Check capacity
capacity_bits = double(pixel_count) * numLSB;
if numel(bits) > capacity_bits
  error('Message too large. Need %d bits but only %d bits available
(numLSB=%d).', ...
      numel(bits), capacity_bits, numLSB);
end
% Embed: replace LSB(s) of pixel vector with message bits
stego_pix = uint8(pix); % copy
bit_idx = 1;
for i = 1 : numel(stego_pix)
```

```
    if bit_idx > numel(bits)
        break
    end
    % Current pixel value
    p = stego_pix(i);
    % For multi-LSB, we replace the lowest numLSB bits with the next bits
chunk
    if numLSB == 1
        stego_pix(i) = bitset(p, 1, bits(bit_idx)); % set bit 1 (LSB)
        bit_idx = bit_idx + 1;
    else
        % get next numLSB bits (pad with zeros if needed)
        chunk = 0;
        for b = 1:numLSB
            if bit_idx <= numel(bits)
                chunk = chunk + bits(bit_idx) * 2^(b-1); % LSB-order insertion
            end
            bit_idx = bit_idx + 1;
        end
        % clear lowest numLSB bits then add chunk
        mask = bitcmp(uint8((2^numLSB)-1));
        stego_pix(i) = bitand(p, mask);
        stego_pix(i) = bitor(stego_pix(i), uint8(chunk));
    end
end
% Reshape back to HxWxC and save
stego_img = reshape(stego_pix, H, W, C);
imwrite(stego_img, output_filename);
fprintf('Embedding done. Wrote %s (used %d bits of %d available).\n',
output_filename, min(numel(bits), capacity_bits), capacity_bits);
```

Extraction Code:

```
stego_filename = 'stego_image.png';
numLSB = 1; % must match embedder
stego = imread(stego_filename);
[H,W,C] = size(stego);
pix = stego(:);% uint8 vector
% Function to read N bits sequentially from pix vector
bit_read = [];
for i = 1 : numel(pix)
  p = pix(i);
  if numLSB == 1
     bit_read(end+1,1) = bitget(p,1); %#ok<SAGROW>
  else
     for b = 1:numLSB
        bit_read(end+1,1) = bitget(p,b);
     end
  end
end
% Read header first: 4 bytes = 32 bits * 8 = 32 bits
header_bits = bit_read(1:32);
% convert bits (MSB-first per byte) into 4 bytes
header_bytes = zeros(4,1,'uint8');
for byte = 1:4
  byte_bits = header_bits((byte-1)*8 + (1:8))';
  header_bytes(byte) = uint8(bin2dec(char(byte_bits+'0')));
end
msg_len = typecast(uint8(header_bytes), 'uint32'); % number of bytes in message
% Now extract message bits: next msg_len*8 bits
start_bit = 32 + 1;
end_bit = 32 + double(msg_len) * 8;
if end_bit > numel(bit_read)
  error('Not enough bits in image to extract the message. Expected %d bits, found
%d.', end_bit, numel(bit_read));
end
```

```
msg_bits = bit_read(start_bit:end_bit);
% group into bytes and convert
msg_bytes = zeros(msg_len,1,'uint8');
for k = 1:double(msg_len)
   chunk = msg_bits((k-1)*8 + (1:8))';
   msg_bytes(k) = uint8(bin2dec(char(chunk + '0')));
end
recovered = char(msg_bytes.');
fprintf('Recovered message (%d bytes): %s\n', msg_len, recovered);
```

OUTPUT:


```
Command Window
New to MATLAB? See resources for Getting Started.
>> embed_lsb
Embedding done. Wrote stego_image.png (used 144 bits of 6220800 available).
>> steg
>> extract_lsb
Recovered message (14 bytes): frame by frame
>>
```

OBSERVATIONS:

1. The secret text is successfully embedded into the colour carrier image using LSB substitution without producing any noticeable visual change.

2. The embedded text can be accurately extracted from the stego image as long as the length of the hidden message is known.

3. The stego image appears visually identical to the original colour image, making it difficult to differentiate between them by normal observation.

4. Only the least significant bits of the RGB channels are modified, which ensures minimal distortion and high imperceptibility.
   .

CONCLUSIONS:

1. The desired text can be hidden inside a colour carrier image effectively using LSB manipulation.

2. The carrier image undergoes very little or no noticeable visual change after embedding the secret text.

3. The method is simple, efficient, and suitable for hiding text or other small data inside digital images.

4. LSB-based steganography proves to be a practical approach for secure data hiding in colour images while maintaining image quality.

2 Image in Image LSB Steganography with XOR Encryption

THEORY:

Image-in-image steganography involves placing one image (secret) into another (carrier) in such a way that the carrier's appearance is almost unchanged. The LSB technique allows modifying only the least significant bit of each pixel to hide data, minimizing visual distortion.

To improve security, the secret image is first encrypted using XOR with a secret key, so that even if someone extracts the LSBs from the carrier, the secret remains unintelligible without the key.

Key points:

1. LSB Embedding: Hides secret image bits in the LSBs of carrier pixels.
2. MSB of Secret: The most significant bit (MSB) of the secret is embedded in the LSB of the carrier to preserve contrast in the recovered image.
3. XOR Encryption: Provides confidentiality by scrambling the secret image before embedding.
4. RGB Channels: Embedding is done separately for Red, Green, and Blue channels.

ALGORITHM:

EMBEDDING ALGORITHM:

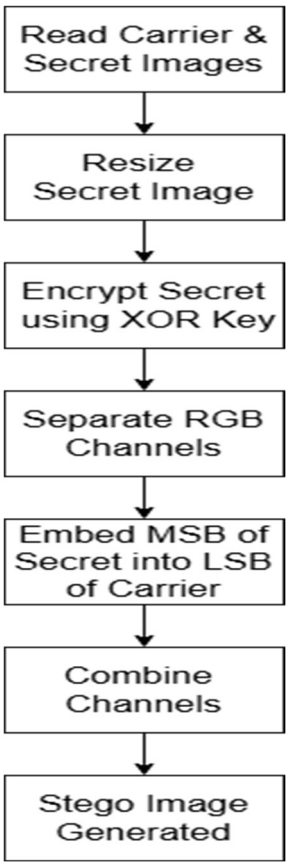
1. Read the carrier and secret images.
2. Resize the secret image to match the carrier dimensions.
3. Encrypt the secret image using XOR with a key (0–255).
4. Separate both images into RGB channels.
5. For each channel:
    I. Clear the LSB of the carrier pixel.
    II. Extract the MSB of the encrypted secret pixel.
    III. Embed it in the LSB of the carrier pixel.
6. Combine the modified RGB channels to form the **stego image**.

EXTRACTION ALGORITHM:

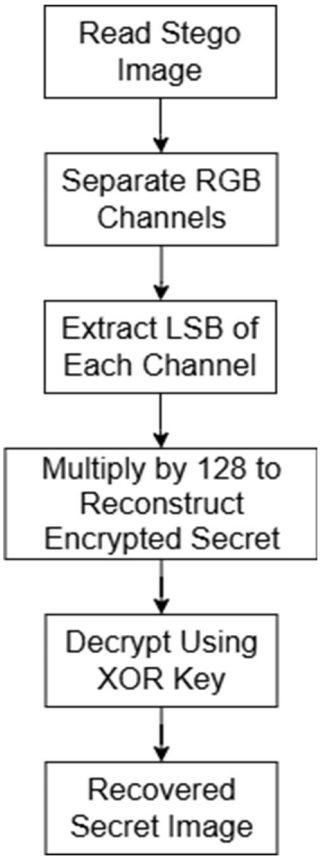
1. Separate the steganographic image into RGB channels.
2. Extract the LSB of each channel, which contains the MSB of the encrypted secret.
3. Multiply the extracted bits by 128 to reconstruct the encrypted secret.
4. Decrypt using XOR with the same key to retrieve the original secret image.

FLOWCHART:

Embedding Flowchart



Extraction Flowchart



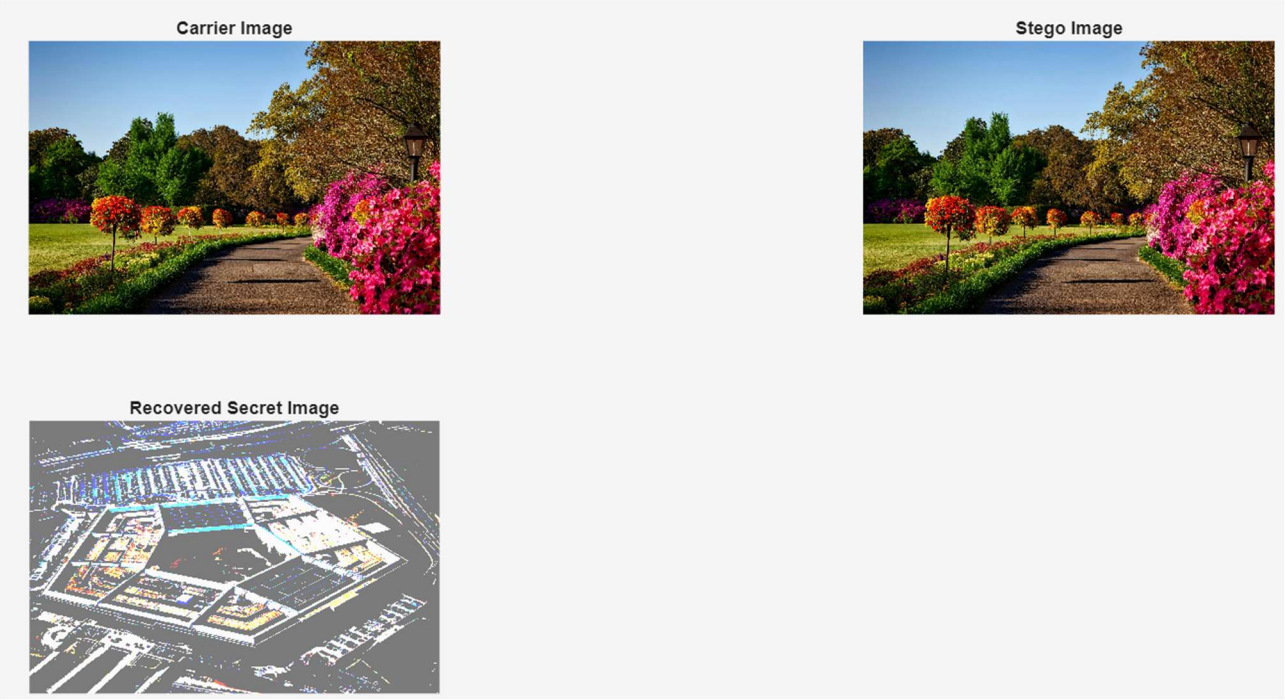
CODE:

```
clc;
clear all;
close all;
% Read the carrier and secret images
carrier = imread('testimage.png');
secret = imread('Secret.png');
subplot(2,2,1);
imshow(carrier);
title('Carrier Image');
% Resize secret image to match carrier size
[r, c, g] = size(carrier);
secret = imresize(secret, [r c]);
% ---- Encrypt the secret image using XOR with a key ----
key = uint8(123); % Choose any number between 0-255 as key
encryptedSecret = bitxor(secret, key);
% Separate RGB channels
ra = carrier(:,:,1);
ga = carrier(:,:,2);
ba = carrier(:,:,3);
rx = encryptedSecret(:,:,1);
gx = encryptedSecret(:,:,2);
bx = encryptedSecret(:,:,3);
% Embed secret bits into carrier image (1-LSB Steganography)
for i = 1:r
  for j = 1:c
    fr(i,j) = bitand(ra(i,j), 254) + bitshift(bitand(rx(i,j),128), -7);
  end
end
redsteg = fr;
for i = 1:r
  for j = 1:c
    fr(i,j) = bitand(ga(i,j), 254) + bitshift(bitand(gx(i,j),128), -7);
  end
```

```
end
greensteg = fr;
for i = 1:r
  for j = 1:c
    fr(i,j) = bitand(ba(i,j), 254) + bitshift(bitand(bx(i,j),128), -7);
  end
end
bluesteg = fr;
% Combine all stego channels
stegoImage = cat(3, redsteg, greensteg, bluesteg);
subplot(2,2,2);
imshow(stegoImage);
title('Stego Image');
% ---- Extract the hidden image ----
redSteg = stegoImage(:,:,1);
greenSteg = stegoImage(:,:,2);
blueSteg = stegoImage(:,:,3);
for i = 1:r
  for j = 1:c
    ms(i,j) = bitand(redSteg(i,j), 1) * 128;
  end
end
recoveredR = ms;
for i = 1:r
  for j = 1:c
    ms(i,j) = bitand(greenSteg(i,j), 1) * 128;
  end
end
recoveredG = ms;
for i = 1:r
  for j = 1:c
    ms(i,j) = bitand(blueSteg(i,j), 1) * 128;
  end
end
```

```
recoveredB = ms;
recoveredEncrypted = cat(3, recoveredR, recoveredG, recoveredB);
% ---- Decrypt the recovered secret image using the same key ----
recoveredSecret = bitxor(recoveredEncrypted, key);
subplot(2,2,3);
imshow(recoveredSecret);
title('Recovered Secret Image');
```

OUTPUT:



ORIGINAL SECRET IMAGE:

OBSERVATIONS:

1. Stego image resembles carrier image.
2. XOR adds security preventing easy access to secret image.
3. There is change in quality of recovered image, suggesting scope for improvement.
4. For each channel, 1 bit is modified to minimize distortion.

CONCLUSIONS:

1. Combined LSB Steganography with XOR Encryption.
2. Simple method yet effective.
3. Technique can be extended for other digital media.

INDIVIDUAL CONTRIBUTION:

1(a): - Sauradipta Basu

1(b): - Shreyas B

2: - Siddharth Bhat

REFERENCES:

https://www.geeksforgeeks.org/computer-graphics/lsb-based-image-steganography-using-matlab/

https://in.mathworks.com/matlabcentral/fileexchange/41326-steganography-using-lsb-substitution?requestedDomain=

https://www.youtube.com/watch?v=ZXfNhSlXT0w

https://www.youtube.com/watch?v=e3ccQogNh70