# Code and application of LSB steganography

# First, we need to install the OpenCV library. You can install it using pip:

EditFull ScreenCopy code

1pip install opencv-python

Now, let's create a function to encode a message into an image:

python

EditFull ScreenCopy code

1import cv2

2import numpy as np

3

4def encode\_text\_in\_image(image\_path, message, output\_path):

5 # Load the image

6 image = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

7

8 # Convert the message to binary data

9 binary\_message = ' '.join(format(ord(i), '08b') for i in message)

10

11 # Get the image dimensions

12 (h, w, \_) = image.shape

13

14 # Initialize the encoded image

15 encoded\_image = image.copy()

16

17 # Initialize the index of the current pixel

18 i = 0

19

20 # Loop over the image and encode the message

21 for y in range(h):

22 for x in range(w):

23 # Get the current pixel

24 pixel = encoded\_image[y, x]

25

26 # Check if we can encode the next bit of the message

27 if i < len(binary\_message):

28 # Get the current pixel value

29 pixel\_value = pixel[2]

30

31 # Get the next bit of the message

32 bit = binary\_message[i]

33

34 # Check if the bit is 1

35 if bit == '1':

36 # Set the least significant bit of the pixel value to 1

37 pixel\_value = pixel\_value + 1

38

39 # Check if the pixel value is odd

40 if pixel\_value % 2 == 1:

41 # Set the least significant bit of the pixel value to 0

42 pixel\_value = pixel\_value - 1

43

44 # Set the pixel value to the new value

45 pixel[2] = pixel\_value

46

47 # Increment the index of the current pixel

48 i += 1

49

50 # If we have encoded all the bits of the message, break the loop

51 if i == len(binary\_message):

52 break

53

54 # Save the encoded image

55 cv2.imwrite(output\_path, encoded\_image)

56

57# Example usage

58image\_path = 'input.png'

59message = 'Hello, World!'

60output\_path = 'output.png'

61

62encode\_text\_in\_image(image\_path, message, output\_path)

This function loads an image, converts the message to binary data, and then loops over the image and encodes the message in the least significant bit of the pixel values. The function saves the encoded image to the specified output path.

To decode the message from the image, we can use the following function:

python

EditFull ScreenCopy code

1def decode\_text\_from\_image(image\_path):

2 # Load the image

3 image = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

4

5 # Initialize the decoded message

6 decoded\_message = ''

7

8 # Get the image dimensions

9 (h, w, \_) = image.shape

10

11 # Loop over the image and decode the message

12 for y in range(h):

13 for x in range(w):

14 # Get the current pixel

15 pixel = image[y, x]

16

17 # Get the least significant bit of the pixel value

18 bit = pixel[2] % 2

19

20 # Convert the bit to a character

21 decoded\_message += chr(int(bit))

22

23 # Return the decoded message

24 return decoded\_message

25

26# Example usage

27image\_path = 'output.png'

28message = decode\_text\_from\_image(image\_path)

29print(message)

This function loads an image and loops over the image to decode the message from the least significant bit of the pixel values. The function returns the decoded message.

Note that this is a basic example of LSB steganography. In practice, you may need to handle more complex cases, such as handling pixel values that are already odd or handling alpha channels.