***Introduction :***

Stenography is not a new or modern technique. Painters and artists across the globe have made use of this technique to conceal signatures and other hidden messages within their art or paintings.

Today, digital steganography is the most common way of concealing information from third parties.

Image Steganography refers to the process of hiding data within an image file. The image selected for this purpose is called the cover image and the image obtained after steganography is called the stego image.

***2. System requirements:***

***2.1. Hardware requirements:***

***2.2. Software requirements:***

***3. Project overview :***

***3.1 Requirements:***

**Input images:**

*Hiding image :*

The decoding takes an image with another image encoded inside as the input.



*Image that is to hidden:*

The decoded output image is identical to the image that was hidden and encoded in another image.

***3.2 Description :***

You hide an image inside another image by encoding it into the least significant bit positions of a subset of pixels in another image. The encoded image can be afterwards decoded and recovered without any information loss.

***4. Project code :***

***4.1 Source code:***

***Encoding :***

* Opens two specified images, an image we want to conceal and an image we want to use for concealing,
* Hides the image information in the binary pixel values of the other image and saves the resulting image in a specified location or the default location if no location is specified.
* The number of pixels in the image used for hiding an image must be at least (2 \* number of pixels in the image to be hidden + 1)

from PIL import Image

from modules import rgb\_to\_binary, add\_leading\_zeros

def main():

img\_visible\_path = input("Enter path of hide-into image : ")

img\_hidden\_path = input("Enter path of image that to be hidden: ")

output\_path = 'img/encoded\_image.png'

img\_visible = Image.open(img\_visible\_path)

img\_hidden = Image.open(img\_hidden\_path)

encoded\_image = encode(img\_visible, img\_hidden) #funtion call

encoded\_image.save(output\_path)

def get\_binary\_pixel\_values(img, width, height):

hidden\_image\_pixels = ''

for col in range(width):

for row in range(height):

pixel = img[col, row]

r = pixel[0]

g = pixel[1]

b = pixel[2]

r\_binary, g\_binary, b\_binary = rgb\_to\_binary(r, g, b)

hidden\_image\_pixels += r\_binary + g\_binary + b\_binary

return hidden\_image\_pixels

def change\_binary\_values(img\_visible, hidden\_image\_pixels, width\_visible, height\_visible, width\_hidden, height\_hidden):

idx = 0

for col in range(width\_visible):

for row in range(height\_visible):

if row == 0 and col == 0:

width\_hidden\_binary = add\_leading\_zeros(bin(width\_hidden)[2:], 12)

height\_hidden\_binary = add\_leading\_zeros(bin(height\_hidden)[2:], 12)

w\_h\_binary = width\_hidden\_binary + height\_hidden\_binary

img\_visible[col, row] = (int(w\_h\_binary[0:8], 2), int(w\_h\_binary[8:16], 2), int(w\_h\_binary[16:24], 2))

continue

r, g, b = img\_visible[col, row]

r\_binary, g\_binary, b\_binary = rgb\_to\_binary(r, g, b)

r\_binary = r\_binary[0:4] + hidden\_image\_pixels[idx:idx+4]

g\_binary = g\_binary[0:4] + hidden\_image\_pixels[idx+4:idx+8]

b\_binary = b\_binary[0:4] + hidden\_image\_pixels[idx+8:idx+12]

idx += 12

img\_visible[col, row] = (int(r\_binary, 2), int(g\_binary, 2), int(b\_binary, 2))

if idx >= len(hidden\_image\_pixels):

return img\_visible

return img\_visible

def encode(img\_visible, img\_hidden):

encoded\_image = img\_visible.load()

img\_hidden\_copy = img\_hidden.load()

width\_visible, height\_visible = img\_visible.size

width\_hidden, height\_hidden = img\_hidden.size

hidden\_image\_pixels = get\_binary\_pixel\_values(img\_hidden\_copy, width\_hidden, height\_hidden)

encoded\_image = change\_binary\_values(encoded\_image, hidden\_image\_pixels, width\_visible, height\_visible, width\_hidden, height\_hidden)

return img\_visible

***Decoding :***

* Opens an image which contains information of a hidden image, recovers the hidden image and saves it in a specified or default location.
* Extracts a sequence of bits representing a sequence of binary values of all pixels of the hidden image.
* The information representing a hidden image is stored in the 4 least significant bits of a subset of pixels of the visible image.

from PIL import Image

from modules import rgb\_to\_binary

def main():

img\_path = input("Enter encoded image path: ")

output\_path = 'img/decoded\_image.png'

decoded\_image = decode(Image.open(img\_path)) #funtion call

decoded\_image.save(output\_path)

def extract\_hidden\_pixels(image, width\_visible, height\_visible, pixel\_count):

hidden\_image\_pixels = ''

idx = 0

for col in range(width\_visible):

for row in range(height\_visible):

if row == 0 and col == 0:

continue

r, g, b = image[col, row]

r\_binary, g\_binary, b\_binary = rgb\_to\_binary(r, g, b)

hidden\_image\_pixels += r\_binary[4:8] + g\_binary[4:8] + b\_binary[4:8]

if idx >= pixel\_count \* 2:

return hidden\_image\_pixels

return hidden\_image\_pixels

def reconstruct\_image(image\_pixels, width, height):

image = Image.new("RGB", (width, height))

image\_copy = image.load()

idx = 0

for col in range(width):

for row in range(height):

r\_binary = image\_pixels[idx:idx+8]

g\_binary = image\_pixels[idx+8:idx+16]

b\_binary = image\_pixels[idx+16:idx+24]

image\_copy[col, row] = (int(r\_binary, 2), int(g\_binary, 2), int(b\_binary, 2))

idx += 24

return image

def decode(image):

image\_copy = image.load()

width\_visible, height\_visible = image.size

r, g, b = image\_copy[0, 0]

r\_binary, g\_binary, b\_binary = rgb\_to\_binary(r, g, b)

w\_h\_binary = r\_binary + g\_binary + b\_binary

width\_hidden = int(w\_h\_binary[0:12], 2)

height\_hidden = int(w\_h\_binary[12:24], 2)

pixel\_count = width\_hidden \* height\_hidden

hidden\_image\_pixels = extract\_hidden\_pixels(image\_copy, width\_visible, height\_visible, pixel\_count)

decoded\_image = reconstruct\_image(hidden\_image\_pixels, width\_hidden, height\_hidden)

return decoded\_image

***4.2 Output :***

Encoded image :

Decoded image :

***5.Application of image stenography :***

* Defense organisation: security from enemies
* Intelligence Agencies: security of person’s private information
* Government Agencies: store critical data like criminal record
* Smart identity cards: personal information is embedded into photo
* Medical: patient’s details are embedded within image

|  |  |  |
| --- | --- | --- |
| Technique | Advantages | Disadvantages |
| Least Signifcant Bit (LSB) Encoding | Hard to detect.  Original image is very similar to altered image.  Embedded data resembles Gaussian noise. | Message is hard to recover if image is subject to attack such as translation and rotation. |
| Low Frequency Encoding | Hard to detect as message and fundamental image data share same range. | Significant damage to picture appearance.  Message difficult to recover. |
| Mid Frequency Encoding | Altered picture closely resembles original.  Not susceptible to attacks such as rotation and translation. | Relatively easy to detect, as our project has shown. |
| High Frequency Domain Encoding | None | Image is distorted.  Message easily lost if picture subject to compression such as JPEG. |

6. Advantages and disadvantages of image stenography :

***7.Conclusion :***

Thus, this encrypting technique helps to send a image with a secret image inside it to somebody.

If done correctly this is undetectable to the naked eye and can be difficult to detect even using the right tools.

For even more security, the secret data can (and should) be encrypted first.

This technique has allegedly been used by Al Qaeda to pass secret messages.

***8.References :***

1. <https://towardsdatascience.com/steganography-hiding-an-image-inside-another-77ca66b2acb1>
2. <https://hackaday.com/2022/05/01/how-to-hide-a-photo-in-a-photo/>
3. <https://incoherency.co.uk/image-steganography/>