Image Steganography in the Frequency Domain

Frederik Imanuel Louis - 13520163
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13520163@std.stei.itb.ac.id

***Abstract*—Image steganography is an important tool used to hide sensitive data in images in order to avoid detection by untrusted parties. This can be done effectively by hiding the data in the frequency domain of the image, instead of directly embedding the data inside of the spatial domain of the image.**

Keywords—DFT, FFT, DCT, Steganography, Embedding

I. Introduction

In this information era, communication and data transmission plays a significant role in our day to day lives. However, this communication carries with it some risks wherein malicious actors may attempt to steal or modify transmitted data. In order to protect this, encryption techniques are needed.

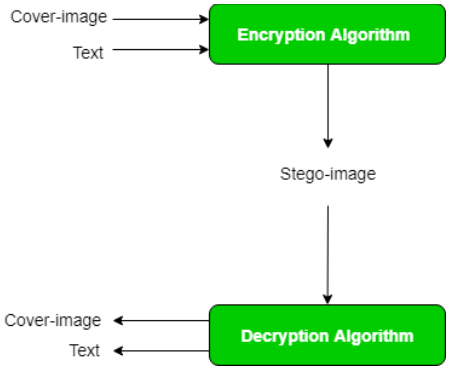
Encryption is a cryptographic technique that aims to protect information so that it cannot be read or understood by unauthorized individuals. These techniques however, does not hide their usage. It is very clear when a data is encrypted and when it is not.

Despite the effectiveness of encryption in safeguarding information, its conspicuous nature may inadvertently attract unwanted attention. The mere presence of encrypted data can invite malicious actors to focus their efforts on decryption attempts. Steganography, on the other hand, introduces a subtler approach to secure communication by concealing the very existence of hidden data within innocuous carriers.

As information technology gets more and more advances, so does attacks against data encryption. Hackers and other malicious actors have found more and more creative ways to attack and recover encrypted data where present. Thus, there arises the need of another technique to conceal the very existence of data transmission, or in other words, methods to transmit data secretly.

Steganography is another cryptographic technique which attempts to hide data in other forms of transmission such that it is hidden. The goal of steganography differs from encryption, where encryption wants to protect data being transmitted such that malicious actors cannot alter or steal the data, steganography strives to hide any indication that data is being sent. Image steganography, for example, attempts to hide data by embedding it into an image.

Furthermore, these two methods are sometimes combined, by first encrypting the data then hiding in using steganography. This means that even if a malicious attacker somehow finds data in an image, the data may still be safe as it is encrypted.



This paper will discuss in detail how image steganography can be done in the frequency domain with two main methods, the Discrete Cosine Transform and the Discrete Fourier Transform, and comparing the results with conventional image steganography.

II. Theoretical Background

A. Image Formats

An image is a two-dimensional grid of pixels, where each pixel represents the smallest unit of visual information. These grids are very small and thousands or even millions of them form the screen of our phone, laptop, or personal computers. In this information era, absurd amounts of images are being sent and received every minute, be it from a website, from social media, from messaging apps, and many other channels. Thus, representing image data efficiently and with the best quality is an important aspect of information theory.

The Bitmap (BMP) image format is a straightforward and widely supported file format that represents images as a grid of pixels, without any compression. BMP files are known for their simplicity and directness in encoding pixel data, making them suitable for basic image storage and manipulation. Unlike compressed formats, BMP files contain raw pixel by pixel color information, resulting in larger file sizes but with the advantage of maintaining image quality without any loss whatsoever.

In a BMP file, each pixel is typically represented by one or more bytes, depending on the color depth of the image. Common color depths include 1, 4, 8, 16, 24, or 32 bits per pixel. The 24-bit BMP format is commonly used for true-color images, providing 8 bits for each of the Red, Green, and Blue color channels. The additional 8 bits in a 32-bit BMP format are often used for an alpha channel, indicating the level of transparency.

One of the most common and widely supported image formats is the JPEG (Joint Photographic Experts Group). JPEG utilizes a lossy compression algorithm, which means that some data is discarded during compression to reduce file size. While this results in smaller file sizes, it may lead to a loss of image quality, particularly in high-compression scenarios. JPEG is well-suited for photographs and images where a slight loss of detail is acceptable, for example in website icons or button images, where efficiency in loading the website is sometimes more crucial than the quality of the images sent.

In contrast, the PNG (Portable Network Graphics) format employs lossless compression, preserving all the original image data. This makes PNG ideal for images that require high fidelity, such as logos, graphics, or images with transparency. However, like the BMP file format, there is a trade-off of larger file sizes compared to JPEG.

Another widely used format is GIF (Graphics Interchange Format), known for its support of animations through a sequence of images. GIF uses lossless compression but has a limited color palette, making it suitable for simple graphics and animations but less so for detailed photographs.

The TIFF or Tagged Image File Format, on the other hand, is a versatile format often used in professional settings. It supports both lossless and lossy compression, and its flexibility allows for the inclusion of multiple images, layers, and metadata.

B. Discrete Cosine Transform

The Discrete Cosine Transform (DCT) is a mathematical transformation widely used in signal processing and image compression. It belongs to the family of Fourier-related transforms and is particularly renowned for its effectiveness in representing signals in terms of a set of cosine functions.

The DCT operates on a finite sequence of data points and transforms it into another set of coefficients, which represent the signal's frequency components. In the context of image compression, the DCT is extensively employed to convert spatial information (pixel values) into frequency information, allowing for a more efficient representation of the image data.

The DCT is especially associated with image and video compression standards such as JPEG (Joint Photographic Experts Group) and MPEG (Moving Picture Experts Group). In these standards, the DCT is applied to non-overlapping blocks of an image, converting spatial information into frequency information for each block. The transformation is reversible, meaning that the original signal (or image) can be accurately reconstructed from its DCT coefficients.

The DCT formula is as follows:

$$C(u) = a(u)\sum_{x=0}^{N-1} f(x)\cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

$$u = 0,1,\ldots,N-1$$

$$a(u) = \begin{cases} \sqrt{\frac{1}{N}} & u = 0 \\ \sqrt{\frac{2}{N}} & u = 1,\ldots,N-1 \end{cases}$$

And the inverse DCT formula is as follows:

$$f(x) = \sum_{u=0}^{N-1} a(u)C(u)\cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

As seen from the formula, DCT does its calculation entirely in the real domain, and is thus reasonably fast to calculate. In JPEG compression, these coefficients are quantized to reduce the precision of the high-frequency components. This quantization step is a key factor in achieving compression, as it allows for more efficient representation of the image information.

The quantized DCT coefficients are then arranged in a zigzag order before being encoded and stored. This zigzag ordering is done because commonly, values closer to the lower right corner in a quantized DCT matrix are comprised entirely of zeros, and this facilitates run-length encoding quite well, a process that groups repeated values for more efficient compression.

C. Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is another mathematical transformat on used in signal processing, similar to the Discrete Cosine Transform (DCT). While the DCT is particularly effective in image and video compression, the DFT is more general-purpose and is widely applied in various domains, including signal analysis, communications, and audio processing.

DFT is extensively used to analyze the frequency content of signals. By transforming a signal from the time domain to the frequency domain, one can identify the presence and intensity of different frequency components. This is crucial in fields such as telecommunications, audio processing, and vibration analysis.

In signal processing, the DFT is employed for filtering applications. Filtering in the frequency domain allows for the removal or attenuation of specific frequency components from a signal. This is useful for tasks such as noise reduction and signal enhancement.

Similar to its application in signal processing, the DFT is used in image processing for tasks such as spatial filtering and image enhancement. Transforming an image to the frequency domain allows for the application of filters to specific frequency components.

In computer vision and image analysis, the DFT is applied to extract features from images. It helps represent an image in a way that emphasizes important patterns or structures, making it useful in tasks like object recognition, such as seperating high frequency colors from low frequency ones.

The DFT formula is as follows:

$$F_{u,v} = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f_{x,y} e^{-i2\pi(ux/N+vy/M)}$$

And the inverse DFT formula is as follows:

$$f_{x,y} = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F_{u,v} e^{i2\pi(ux/N+vy/M)}$$

Notably from the formula, DFT does its calculation in the real and imaginary domain. Thus, this calculation is more expensive than the calculation used and JPEG, and is not used in representing images.

D. Image Steganography

Image steganography is the technique of concealing information within an image, allowing for covert communication between parties in order to avoid detection. Unlike encryption, which transforms the original information into an unreadable format, steganography focuses on hiding the existence of the communicated data. The primary goal is to embed secret messages within an image in such a way that the alterations are imperceptible to the human eye and difficult for automated systems to detect.

The roots of steganography can be traced back to ancient times, where secret messages were concealed within wax tablets or tattooed onto the scalps of messengers. However, the digital era has ushered in new possibilities for steganography, especially within the realm of multimedia, where images provide an excellent medium for embedding hidden information.

The modern history of image steganography can be linked to the early days of digital communication when researchers sought ways to secure data transmission without raising suspicion. Over the years, as digital images became commonplace, the focus shifted to hiding information within these visual artifacts.

Least Significant Bit (LSB) steganography is a widely employed and conceptually straightforward method used in image steganography. This technique involves the replacement of the least significant bits of pixel values with the secret data. Since these bits contribute minimally to the overall intensity of a pixel, such alterations are imperceptible to the human eye, making LSB steganography an effective and low-impact method for concealing information within digital images.

For example, this is a comparison of an image that been embedded with a secret message (right) and the original unaltered image (left):

To the naked eye, they look entirely identical. However, as this LSB method has become extremely common, tools to detect this has also become increasingly stronger. One simple yet powerful example is Zsteg.

Zsteg is a command-line tool designed for steganography analysis in PNG and BMP images. Steganography involves hiding information within other non-secret data, such as images, audio files, or text. The zsteg tool, specifically, is focused on extracting hidden data from the least significant bits of pixel values in images.

The following is an example usage of zsteg in use on an image from a cyber security contest:

```
imagedata           .. text: "Uhv4BR$.<!(2#(0\"(2#*4*4@;L\\l"
b1,b,lsb,xy         .. text: "flag{0rc45_4r3nt_6lu3_s1lly_4895131}"
b2,r,lsb,xy         .. text: "UUTFUUUW"
b2,g,lsb,xy         .. file: PGP Secret Key -
b2,g,msb,xy         .. text: "iiUueUUu"
b2,b,msb,xy         .. text: "TUUUEUTAUE"
b2,bgr,msb,xy       .. text: "VU_UUUUeY"
b4,r,lsb,xy         .. text: "UEUffufffgUfffeeEfUVVuVeUUfVffUUVfvvDffVfUUW
UUfgffeVeVgfffUUUUUUUUUUVehveUfUVfegeUUUUVgeUVggeUUUUUUUUUUUeVgvVUUUvUEUV
b4,r,msb,xy         .. text: "jfnn\"ffjf"
b4,g,lsb,xy         .. text: "\"\"\"\"\"\"2"
b4,g,msb,xy         .. text: "DHDDDDDDDL"
b4,b,msb,xy         .. text: "7swwwww73sw337s73swww7swwww7w7773www77sww7sw
```

It can automatically and very easily analyze all possible bit channels of the image, such as the i-th LSB of every color channel, and try to extract text from each of them.

III. Frequency Based Steganography

A. DFT Steganography Implementation

The DFT image steganography will insert a phrase into the red channel of an RGB image. This works for images in any format, but preferably, the output image is saved using a lossless image format. The following is the DFT embedding implementation in MATLAB.

```
filename = "D:\Github\image-
restoration\images\azusachibi.jpg";
message = "secret";
```

```
data = double(char(message));
data = [data, 0, 0, 0, 0];
im = imread(filename);
imshow(im);
red = im(:, :, 1);
arr = fft2(red);
compl = imag(arr);
arr = real(arr);
r = size(arr, 1);
c = size(arr, 2);
for i = 1:size(data, 2)
    arr(r, c) = data(1, i);
    c = c - 1;
    if c == 0
        c = size(arr, 2);
        r = r - 1;
    end
end
newfftred = arr + 1i * compl;
newred = real(ifft2(newfftred));
newim = im;
newim(:, :, 1) = newred;
imshow(newim);
```

The implementation inserts the ASCII values of the phrase into the last row of the FFT transformed red channel of the image. Unlike the LSB method, there is no need to embed the ASCII as binary, bit per bit, into each pixel value. This is because the "bottom" part of the frequency domain will show up less often, and because of that, embedding the raw values will still not be noticeable to the naked eye. The ASCII values will only be inserted into the real part of the FFT transform, and the imaginary part will be restored after the embedding process is complete.

The following is some comparisons of the original image (left) and the embedded image (right):

Notice that even when embedding ASCII values directly without splitting it bit by bit, the embedded image still looks fine at a glance.

B. DCT Steganography Implementation

The DCT image steganography will insert a phrase into the red channel of an RGB image, similar to the DFT implementation. This works for images in any format, but preferably, the output image is saved using a lossless image format. The following is the DCT embedding implementation in MATLAB.

```
filename = "D:\Github\frequency-
steganography\images\pemaloe.png";
message = "secret";
data = double(char(message));
data = [data, 0, 0, 0, 0];
im = imread(filename);
imshow(im);
red = im(:, :, 1);
arr = dct2(red);
r = size(arr, 1);
c = size(arr, 2);
for i = 1:size(data, 2)
    arr(r, c) = data(1, i);
    c = c - 1;
    if c == 0
        c = size(arr, 2);
        r = r - 1;
    end
end
newred = idct2(arr);
newim = im;
newim(:, :, 1) = newred;
imshow(newim);
```

The implementation inserts the ASCII values of the phrase into the last row of the DCT transformed red channel of the image. Similar to the FFT method, there is no need to embed the ASCII as binary, bit per bit, into each pixel value. This is because the "bottom" part of the frequency domain will show up less often, and because of that, embedding the raw values will still not be noticeable to the naked eye. After embedding, the

values will be transformed back into the spatial domain and saved as an image.

The following is some comparisons of the original image (left) and the embedded image (right):



Similar to the FFT embedding, the difference of the embedded image and the original image is not noticeable to the human eye.

IV. Embedding Analysis

First, we will try to detect for any anomalies from the embedded image using Zsteg.

```
$ ls
azusachibi.jpg azusachibi_dct.bmp azusachibi_fft.bmp pemaloe.png pemaloe_dct.bmp pemaloe_fft.bmp
$ zsteg azusachibi_fft.bmp
imagedata                .. text: "A<mG@gA8WUNeE@Q>;ESQX%"
b3,r,lsb,xY              .. file: OpenPGP Public Key
b4,r,lsb,xY              .. text: "`Pxu oPH"
b4,r,msb,xY              .. text: "qYJ&JKYX4"
$ zsteg azusachibi_dct.bmp
imagedata                .. text: "A<[G@UA8EUNSE@?>;3SQF%"
b2,msb,bY                .. file: OpenPGP Public Key
$ zsteg pemaloe_fft.bmp
imagedata                .. text: "C \"N0)T9,[=-]@1Z;.O/)?!!9"
b4,r,msb,xY              .. text: ["3" repeated 9 times]
$ zsteg pemaloe_dct.bmp
imagedata                .. text: "!*\n!(\n! "
[=] nothing :(
```

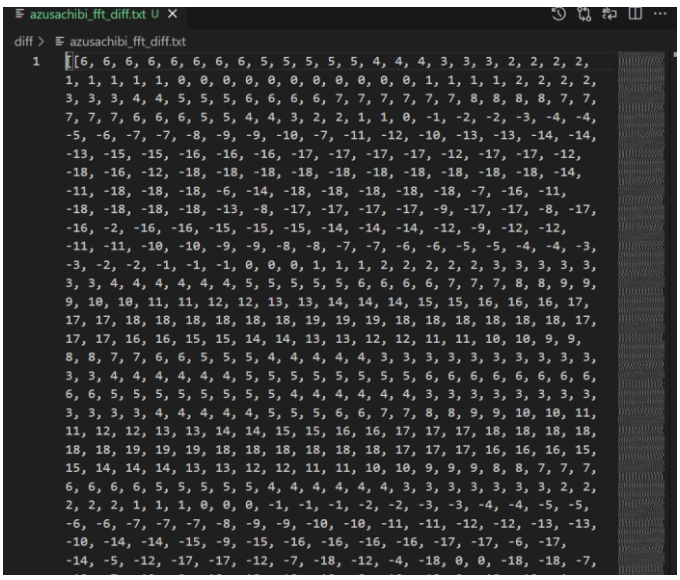
We see that Zsteg is unable to find our embedded secret in any of the images. Next, we will check the byte difference of the original image and the embedded image. As we only modified the red channel, it will be the only channel that we analyze.

We will use the following Python code to analyze the difference:

```python
from PIL import Image
import numpy as np
original =
Image.open('./images/azusachibi.jpg')
fft =
Image.open('./images/azusachibi_fft.bmp')
original = np.array(original)
fft = np.array(fft)

diff = []
for row_ori, row_fft in zip(original,
fft):
    diff.append([])
    for pixel_ori, pixel_fft in
zip(row_ori, row_fft):
        diff[-1].append(int(pixel_ori[0])
- int(pixel_fft[0]))
open('./diff/azusachibi_fft_diff.txt',
'w+').write(str(diff))
```

The following is a snippet of the difference of the FFT embedded image and the original image:



We see that even though we embedded the images using full bytes directly into the FFT transformed dimension, the byte difference on the red channel is distributed somewhat evenly over the entire image, with small differences of the pixel values. This method is thus harder to detect than embedding methods that directly use the spatial domain.

V. Conclusion

Frequency embedding works relatively well to hide secret data into images. By transforming images into the frequency domain first, then embedding the secret there, the secret is made harder to detect as the byte difference created when embedding the secret is spread evenly when transforming the image back to the spatial domain

Moving forward, the embedding method may be improved by encrypting the data first before embedding it. Other than that, it may also be improved by using the imaginary part of the FFT frequency domain transformed image better, either to hide more data or hide the tampering better.

Code Repository

The implementation of the frequency image embedding and its analysis can be accessed at Github using the following link: https://github.com/dxt99/frequency-steganography

Acknowledgment

First of all, I thank God for giving me the chance to research and write about this fascinating topic. I also thank the lecturer of IF4073 Interpretasi dan Pengolahan Citra, Dr. Ir. Rinaldi Munir, M.T., that have encouranged and guided us to write this paper.

Statement

I, Frederik Imanuel Louis, hereby declare that this paper is written originally by myself, and is not a translation, a copy, or plagirized from any sources

Bandung, December 18 2023

Frederik Imanuel Louis

References

[1] Munir, Slide Kuliah IF4073 Interpretasi dan Pengolahan Citra Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2023-2024/citra23-24.htm

[2] Zsteg, Available: https://github.com/zed-0xff/zsteg

[3] Munir, Steganografi dari Kuliah IF4020 Kriptografi, Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/08-Steganografi-Bagian1-2023.pdf