Implementation of Fractal Image Compression in Image Steganography Utilizing LSB Technique

Edbert Eddyson Gunawan - 13522039[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522039@mahasiswa.itb.ac.id

***Abstract*— With the vast development of technology, the needs for enhanced image concealment and secure communication have increased. Fractal compression is renowned for its ability to efficiently compress images while maintaining visual quality. In this paper, Fractal Image Compression is employed to efficiently reduce the space required for storing images, while LSB method is used for concealing data within the least significant bits of pixel values. The integration of these methods capitalizes on their respective strengths, providing a comprehensive solution for secure and space-efficient information hiding. This paper investigates the implementation details, including the fractal compression process, LSB embedding strategy, and the trade-offs involved in terms of image quality, space utilization, and computational complexity. A comparative analysis with traditional LSB methods is conducted to highlight the advantages of the integrated approach. Experimental results demonstrate the effectiveness of the integrated methodology, showcasing reduced storage space requirements and enhanced security.**

***Keywords*—Fractal Compression, LSB, Steganography**

I. INTRODUCTION

With the recent development of computational technology and intrusion system, the needs for entities to have personal communication have increased. Concealing sensitive information within digital images has become a critical area of research and development. The fusion of two powerful techniques, Fractal Compression and Least Significant Bit (LSB) modification, presents an innovative approach to achieve covert communication and secure data hiding. Fractal Compression, known for its ability to efficiently represent complex visual patterns, is integrated with LSB modification, a widely used method for altering the least significant bits of pixel values in digital images.

Fractal Compression, renowned for its ability to represent complex visual patterns through iterative transformations, serves as the foundational framework for our proposed concealment technique. By exploiting the inherent self-similarity found in natural images, fractal compression allows for efficient encoding and decoding of data. This characteristic is leveraged to embed concealed information seamlessly within the structure of digital images, forming the basis of our covert communication strategy.

Complementing the fractal compression framework, this paper incorporate LSB modification, a method frequently employed for inconspicuously altering digital images. LSB modification involves the subtle alteration of the least significant bits of pixel values, which often results in imperceptible changes to the human eye. In our approach, these modified LSBs act as carriers for the concealed information, ensuring that the hidden data remains undetected within the cover image.

The synergistic amalgamation of Fractal Compression and LSB modification offers a multi-layered security paradigm. The intricacies of the fractal representation provide an efficient means of embedding data, while LSB modification imparts a layer of stealth by introducing imperceptible changes to the carrier image. This combination aims to address the challenges of secure communication, where the hidden information remains elusive to visual scrutiny, thus enhancing the overall robustness of our proposed technique.

This paper aims to provide a comprehensive exploration of the implementation of Fractal Compression for image concealment through LSB modification. To check the efficacy of the proposal, we calculated the execution time and the size of cover image needed.

The paper has been organized as follows: Section 2 provides the theoretical framework, Section 3 provides the proposed scheme, Section 4 provides the implementation, Section 5 provides the simulation result, and Section 6 provides the conclusion followed by references.

II. THEORETICAL FRAMEWORK

A. Iterated Function System

Iterated Function System, introduced by Michael Barnsley, is a mathematical construct employed to generate fractals through the repeated application of a set of contractive transformations. In the context of fractal compression, IFS serves as the backbone for capturing and reproducing intricate visual patterns with remarkable efficiency. The fundamental premise involves defining a collection of affine transformations, each contributing to the self-replicating nature of the fractal.

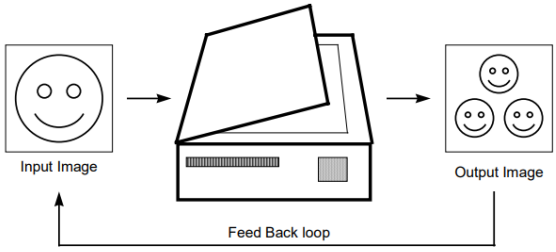

Figure 1. IFS copy machine illustration
Source: Adapted from [1]

The IFS can be imagines as a photocopying machine that takes image as input, then reduce the sized by half, and reproduces it three times on the copy. This process then repeated multiple times as the output image then insert back as the input image of the machine. The resulting image would always be convergent and created a Sierpinski triangle.

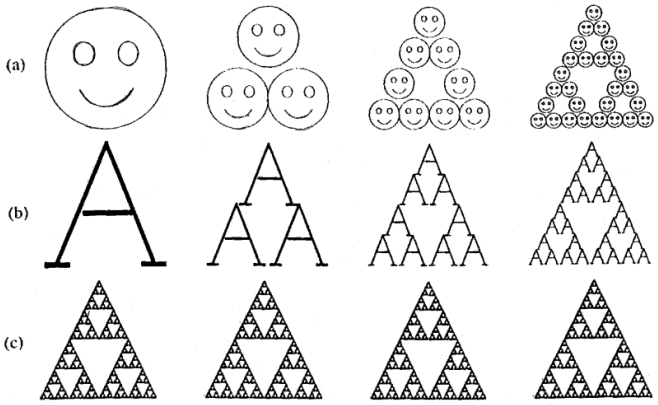

Figure 2. Example of images convergent
Source: Adapted from [1]

The IFS can be seen as a series of affine transformation of the plane. Each can skew, stretch, rotate, scale and translate a plane. Where for any pixels (x, y) in an image then being transformed as

$$w\begin{bmatrix}x\\y\end{bmatrix} = \begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}e\\f\end{bmatrix} \quad (1)$$

As such, each transformation w is defined by 6 numbers $a, b, c, d, e,$ and f. For each

A critical property of the IFS affine transformations is their contraction nature. Each w must be a contraction, ensuring that the distance between points diminishes with each iteration. This contraction property is fundamental for the creation of self-similar structures within the fractal. As iterations progress, the points approach a limiting set, embodying the inherent self-similarity characteristic of fractals.

Two noteworthy theorems are the contraction mapping theorem and the Collage Theorem. The contraction mapping theorem, formally articulated with (X, d) denoting a complete metric space and T: X → X representing a mapping, asserts the Banach Fixed-Point Theorem. This theorem stipulates that if there exists a constant $0 \leq k < 1$ such that, for all x, y ∈ X, the distance between the images of x and y under T adheres to the inequality $d(T(x), T(y)) \leq k \cdot d(x, y)$, then T possesses a unique fixed point in X. The second theorem, the Collage Theorem, informs us that discovering a contraction f such that f(x) is proximate to x ensures that the fixed point of f is likewise close to x.

A shared characteristic among transformations operating in a looping manner is their ability to generate images in which each iteration arises from transformed and diminished replicas of the original. Consequently, these images exhibit intricate details across multiple scales, adhering to the essence of fractals. This fractal generation technique, credited to John Hutchinson, is further expounded upon in works by Barnsley and Peitgen, Saupe, and Jurgens, providing comprehensive insights into the diverse methodologies employed for creating such fractals.

B. Fractal Image Compression

The main idea of fractal compression lies in identifying the match between the domain blocks and range blocks. The first step to do fractal compression be to partition the images in blocks that are disjoin and cover the whole image, that we called as *range* blocks. This paper used rectangle as the partition shape. Then the original image were being reduced by averaging (down sampling and low pass-filtering) to half the size of the *range* blocks domain, this is what we called as *domain* blocks.

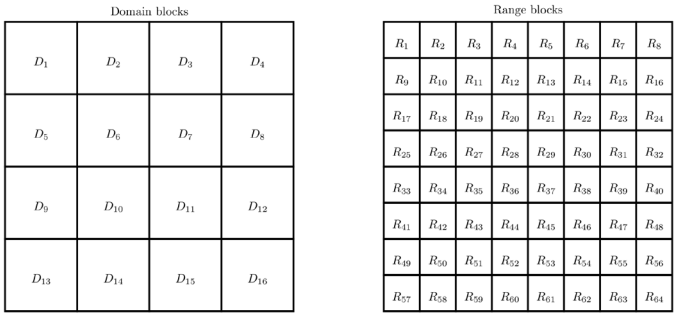

Figure 3. Domain blocks and range blocks
Source: Adapted from [6]

Furthermore, we then define the distance, d, from Frobenius form as

$$d(x,y) = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(x_{ij} - y_{ij}\right)^2} \quad (2)$$

We then perform *affine* transformation to each block, mapping, for each *range* blocks to *domain* blocks. As such the equation would be

$$z' = s \times z + o \quad (3)$$

s, o is defined as contrast and brightness. In cases where s is valued at 0, the pixel becomes dark; when s equals 1, the contrast remains unchanged. Between 0 and 1, the pixel experiences a decrease in contrast, while values exceeding 1 result in an increase in contrast. The parameter o signifies the brightness offset of the pixe. A positive value of o brightens the image, whereas a negative value darkens it.

Each transformed *domain* blocks then compared to each *ranged* blocks in order to find the closest domain blocks for each range. The result would be then stored as the value.

C. Image Steganography

One popular method to conceal image is by using Least Significant Bit (LSB). The common approach to steganography implementation often involves utilizing the Least Significant Bit (LSB) method. The convenience in both embedding and retrieving data has established this method as a frequently employed technique. In essence, this method operates by substituting the least significant bit within the binary representation of the constituent elements of the media. For example, consider the scenario where a digital image measuring $m \times n$ pixels is under consideration, signifying the presence of mn pixels in the image. Each pixel comprises a blend of three colors—red, green, and blue (RGB)—with each color being constructed from 8 bits. The modification of the least significant bit within this 8-bit array is employed for the storage of the intended message. Through this implementation, each pixel can effectively contain 3 bits of confidential information. The

adoption of the LSB method serves the purpose of minimizing discernible differences resulting from message embedding. To illustrate further, imagine an image with the binary code of the initial 3 pixels and LSB 001 000 011 presented as follows:

Table 1. Example of the first 3 pixels of an image

R	G	B
11110110	10101110	10001011
11001100	11111000	10101010
11111110	11111111	11110101

If then the message “111 111 111”concealed in to the pixel, the result would be:

Table 2. Example of the first 3 pixels of an image after LSB performed

R	G	B
11110111	10101111	10001011
11001101	11111001	10101011
11111111	11111111	11110101

III. PROPOSED SCHEME

A. Encryption Scheme

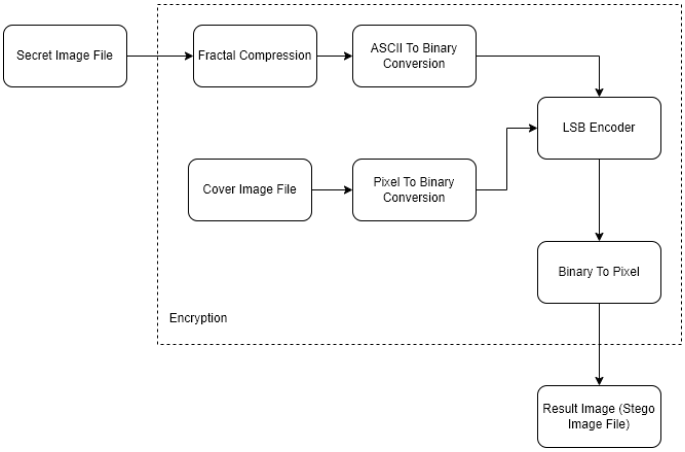

Figure 4. Proposed encryption scheme

The proposed methodology begins with the initial step of acquiring both the secret and cover images. Subsequently, a fractal compression process is applied to the secret image with the source blocks 8x8 and destination block 4x4, wherein the evaluation of distances between domain blocks and range blocks is facilitated by the following formula, where 'd' represents distance,

$$d(x,y) = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(x_{ij} - y_{ij}\right)^2} \qquad (2)$$

Furthermore, the mapping function is characterized by the equation

$$f\left(x_{D_k}\right) = s \times rotate_{\theta}\left(flip_d\left(reduce\left(x_{D_k}\right)\right)\right) + o \qquad (3)$$

incorporating parameters such as contrast 's', brightness 'o', image angle 'θ' [0, 90, 180, 270], and a binary indicator for block flipping [1, -1]. The functions *rotate*, *reduce*, and *flip* are introduced as affine transformations, responsible for rotating the image to a specified angle 'θ', reducing image size, and flipping the image, respectively.

In the subsequent step, the outcome of the fractal compression undergoes binary encoding. Following this, the steganography scheme is implemented by embedding the confidential message into the least significant bit of the cover image utilizing the LSB method. The final step involves the preservation of the resultant image, which is subsequently transmitted to another recipient for further consideration.

B. Decryption Scheme

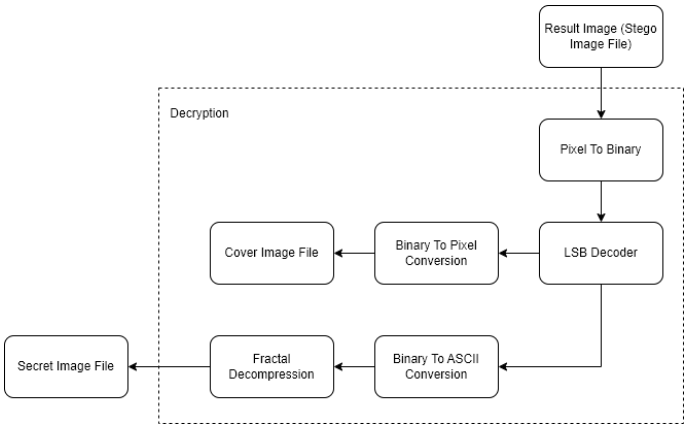

Figure 5. Proposed decryption scheme

The decryption scheme starts with the retrieval of the resultant image obtained from steganography. Subsequently, the pixels are converted into binary form, serving as input for the LSB Decoder. Following this, the resulting binary data undergoes conversion into ASCII format and is subjected to fractal image decompression, employing an 8x8 source block and a 4x4 destination block. The final outcome yields the secret image file.

IV. IMPLEMENTATION

This program is developed utilizing Python as the primary programming language due to its versatility in image processing. The libraries employed include PIL for image processing, *Argparse* for facilitating the command-line interface, *numpy* for numerical processing, *stepic* for executing LSB steganography, and *matplotlib* for image visualization.

Several limitations have been incorporated into this implementation to ensure the program's feasibility. Firstly, the secret image must be in grayscale PNG format, chosen for its ease in pixel manipulation using Python. Secondly, the sizes of both range and domain blocks are predetermined to enable the functioning of fractal compression.

The source code of this program can be found on below links for further developments.

https://github.com/WazeAzure/fractal-lsb-steganography

A. Main Program

```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    subparser = parser.add_subparsers(dest='operation', help='Available operations')

    encrypt_parser = subparser.add_parser('e', help='encrypt image')
    encrypt_parser.add_argument('image_secret', metavar='s_path', help='input file path of image to hide')
    encrypt_parser.add_argument('image_cover', metavar='i_path', help='input file path of image as cover')
    encrypt_parser.add_argument('image_result', metavar='r_path', help='input file path for output image')

    decrypt_parser = subparser.add_parser('d', help='decrypt secret image')
    decrypt_parser.add_argument('image_result', metavar='r_path', help='input file path for output image')
    decrypt_parser.add_argument('image_secret', metavar='s_path', help='input file path of image to hide')

    test_parser = subparser.add_parser('t')
    test_parser.add_argument('image_secret', metavar='s_path', help='input file path of image to hide')

    args = parser.parse_args()
    if args.operation == 'e':
        image_hide = args.image_secret
        image_cover = args.image_cover
        image_result = args.image_result
        start = time.time()
        encrypt(image_hide, image_cover, image_result)
        end = time.time()
        print("Encode Elapsed:", end-start)
    elif args.operation == 'd':
        image_result = args.image_result
        image_hide = args.image_secret
        start = time.time()
        decrypt(image_result, image_hide)
        end = time.time()
        print("Decode Elapsed:", end - start)
```

Figure 6. Main program source code

In this section, all functions within the program are implemented in accordance with the operation and characteristics of each function. This program features a command line interface that enables users to perform encryption and decryption on an image through the application of steganography based on the input and control flow desired by the user. The implementation is outlined in figure 6.

B. Encrypt

```python
def encrypt(image_hide, image_cover, image_result):
    img = cv.imread(image_hide)
    img = get_greyscale_image(img)
    img = img_reduce(img, 4)

    transformations = compress(img, 8, 4, 8)

    secret_msg = str(transformations)
    secret_msg = re.sub(r'[()\[\]]', '', secret_msg)
    secret_msg = secret_msg.encode()

    original_img = Image.open(image_cover)
    original_img = original_img.convert('RGB')

    encoded_img = stepic.encode(original_img, secret_msg)
    encoded_img.save(image_result)
```

Figure 7. Encryption function

This function implements the encryption scheme discussed in the problem analysis. Additionally, it incorporates an additional feature, namely, the execution time, allowing for observation of changes corresponding to each variation.

C. Decrypt

```python
def decrypt(image_result, image_hide):
    secret_msg = Image.open(image_result)
    secret_msg = stepic.decode(secret_msg)

    transformations = str_to_np(secret_msg)

    iterations = decompress(transformations, 8, 4, 8)
    plt.imsave(image_hide, iterations[-1], cmap='gray')
```

Figure 8. Decryption function

This function implements the decryption scheme discussed in the problem analysis. Additionally, it incorporates an additional feature, namely, the execution time, allowing for observation of changes corresponding to each variation.

V. SIMULATION RESULT

In this simulation this simulation, the secret image that being used is *monkey.png* in greyscale. That going to be hide inside of cover image.

Figure 9. Secret image of monkey.png
Source: Adapted from [6]

There would be 2 cover image used *totoro.jpg* and *lena256x256.gif*, with respective size 1050 x 658 pixels and 256 x 256 pixels.

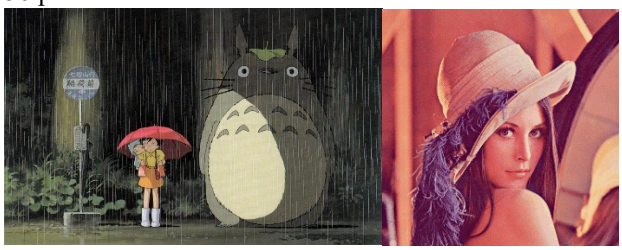

Figure 10. Cover image totoro.jpg and lena256x256.gif
Source: Adapted from https://m.media-amazon.com/images/M/MV5BMTg1NzkyNDk4N15BMl5BanBnXkFtZTgwMDE2MDIyMDE@._V1_.jpg, https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/lena.html

Table 3. File details

Parameters	Filename		
	monkey.png	totoro.jpg	lena256x256.gif
Size (kilo bytes)	40.8	628	62.7
Resolution (px)	256x256	1050x658	256x256
Format	png	jpg	gif

A. Basic Image LSB – lena256x256.gif

Using lena256x256.gif as cover image, it is shown that the program raised error to mentioning that the data to hide is too large for the image *lena256x256x.gif* to contain.

```
PS C:\Users\zakic\Desktop\Makalah Matdis - Edbert> python .\standard_lsb2.py
Traceback (most recent call last):
  File "C:\Users\zakic\Desktop\Makalah Matdis - Edbert\standard_lsb2.py", line 42, in <module>
    encode(img_path, result_path, secret_img)
  File "C:\Users\zakic\Desktop\Makalah Matdis - Edbert\standard_lsb2.py", line 17, in encode
    encoded_img = stepic.encode(original_image, secret_img)
  File "C:\Users\zakic\AppData\Local\Programs\Python\Python310\lib\site-packages\stepic\__init__.py", line 104, in encode
    encode_inplace(image, data)
  File "C:\Users\zakic\AppData\Local\Programs\Python\Python310\lib\site-packages\stepic\__init__.py", line 90, in encode_inplace
    for pixel in encode_imdata(image.getdata(), data):
  File "C:\Users\zakic\AppData\Local\Programs\Python\Python310\lib\site-packages\stepic\__init__.py", line 64, in encode_imdata
    raise ValueError('data is too large for image')
ValueError: data is too large for image
PS C:\Users\zakic\Desktop\Makalah Matdis - Edbert>
```

Figure 11. Encryption process using lena256x256.gif as cover image

This because the LSB from each cover image pixels are not big enough to store the secret image data.

B. Basic Image LSB – totoro.jpg

Figure 12. Result image

```
PS C:\Users\zakic\Desktop\Makalah Matdis - Edbert> python .\standard_lsb2.py
Encode Elapsed: 0.7306075096130371 s
Decode Elapsed: 0.17749476432800293 s
```

Figure 13. Encryption and decryption process

Figure 14. Retrieved secret image

Using totoro.jpg as cover image, it can be seen that the steganography encryption succeed and the result of the image would be seen in the folder.

C. Modified LSB – lena256x256.gif

Figure 15. Result image

```
C:\Users\zakic\Desktop\Makalah Matdis - Edbert>python main.py e images/monkey.png i
mages/lena256x256.gif result/hasil1.png
C:\Users\zakic\Desktop\Makalah Matdis - Edbert\main.py:75: FutureWarning: `rcond` p
arameter will change to the default of machine precision times ``max(M, N)`` where
M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`,
to keep using the old, explicitly pass `rcond=-1`.
  x, _, _, _ = np.linalg.lstsq(A, b)
Encode Elapsed: 8.097744703292847 s

C:\Users\zakic\Desktop\Makalah Matdis - Edbert>
```

Figure 16. Encryption process

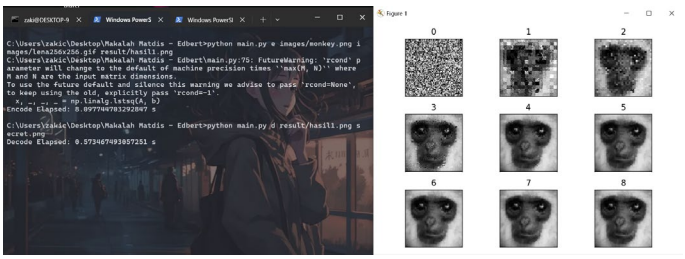

Figure 17. Decryption process

In this section, the program takes argument from command line, then encrypt the resulting the tuple from fractal compression into the cover image. Upon decryption, the program only takes the result image path, output image path then plots the secret image.

D. Modified LSB – totoro.jpg

Figure 18. Result image

```
C:\Users\zakic\Desktop\Makalah Matdis - Edbert>python main.py e images/monkey.png i
mages/totoro.jpg result/hasil1.png
C:\Users\zakic\Desktop\Makalah Matdis - Edbert\main.py:75: FutureWarning: `rcond` p
arameter will change to the default of machine precision times ``max(M, N)`` where
M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`,
to keep using the old, explicitly pass `rcond=-1`.
  x, _, _, _ = np.linalg.lstsq(A, b)
Encode Elapsed: 8.23906922340393 s

C:\Users\zakic\Desktop\Makalah Matdis - Edbert>
```

Figure 19. Encryption process

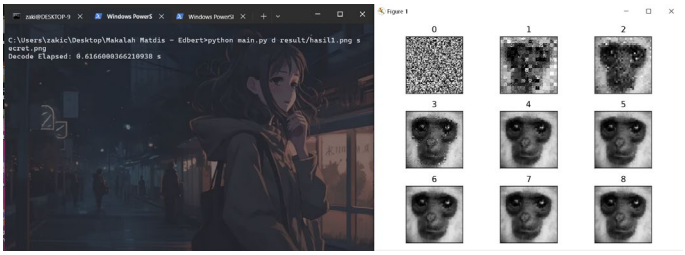

Figure 20. Decryption process

In this section, the program takes argument from command line, then encrypt the resulting the tuple from fractal compression into the cover image. Upon decryption, the program only takes the result image path, output image path then plots the secret image.

E. Simulation Analysis

Table 4. Result of using lena256x256.gif cover image

Parameters	Standard Image LSB	Modified Image LSB
Time Encrypt (s)	X	8.098
Time Decrypt (s)	X	0.573
Result Image Size (kilo bytes)	X	103

Table 5. Result of using totoro.jpg cover image

Parameters	Standard Image LSB	Modified Image LSB
Time Encrypt (s)	0.731	8.239
Time Decrypt (s)	0.177	0.617
Result Image Size (kilo bytes)	1130	1090

Table 4 illustrates that the conventional LSB algorithm faces limitations in storage performance due to the insufficient size of the LSB from the cover image, incapable of accommodating the entirety of the secret image data. Conversely, employing fractal compression enables efficient image preservation, resulting in reduced space consumption, as evident from the smaller size of the resulting image in Table 5. Nevertheless, this improvement comes with a trade-off, namely, an increased encryption and decryption time compared to the basic method. This is attributed to the time-consuming nature of the fractal image compression algorithm, primarily during the partitioning and storage of the fractal representation of the secret image.

Upon decryption, the reconstructed image exhibits slightly diminished quality compared to the original secret image, although the content remains perceptible. Consequently, both compression methods, modified LSB steganography and fractal compression, demonstrate efficacy in concealing an image. Modified LSB steganography offers space efficiency compared to the standard method, albeit with the compromise of increased encryption and decryption times.

A primary advantage of the modified algorithm lies in its capacity to store floating-point numbers. These numbers, when directly extracted without employing fractal decompression, are hard to comprehend. Consequently, this introduces an additional layer of complexity for potential attackers, as they would need to discern the usage of these numbers before decoding them into a visually perceivable image

VI. Conclusion

Steganography serves as a technique for concealing confidential information within alternative media. A common approach involves utilizing the Least Significant Bit (LSB) technique to embed text or images within other images. The integration of fractal compression into steganography methodologies demonstrates its feasibility. The outcomes reveal that fractal compression necessitates a smaller space within the cover image compared to the fundamental LSB algorithm. Additionally, it introduces an added layer of complexity for potential attackers attempting to discern the significance of floating-point numbers embedded in the image. Consequently, without proper identification and decryption, deciphering the secret image into a visually perceivable form becomes challenging. However, it is noteworthy that the encryption and decryption time for the modified LSB algorithm is higher in comparison to the standard approach.

This study lays the groundwork for further development. The current program exclusively encrypts grayscale images, and an enhancement to accommodate RGB images would be beneficial. Furthermore, the program's limitation to certain file types could be broadened, allowing for more extensive file compatibility and thereby improving clandestine communication capabilities.

VII. Acknowledgement

Praises and gratitude are due to the Almighty for His blessings and abundant grace, enabling the author to successfully complete this paper. The author extends sincere thanks to Dr. Nur Ulfa Maulidevi, S.T, M.Sc, the lecturer for the IF2120 Discrete Mathematics course (K01 class), for imparting valuable knowledge, which greatly contributed to the successful completion of this paper. Special thanks are also extended to Dr. Ir. Rinaldi, M.T, for providing insightful advice regarding fractal image compression. Additionally, heartfelt appreciation is conveyed to the author's parents for their unwavering support and motivation throughout the process.

Reference

[1] Texas Instrument, "An Introduction to Fractal Image Compression", 1997
[2] Welstead, Stephen. "Fractal and Wavelet Image Compression Techniques", SPIE Publications, 1999.
[3] Barnsley, Micahel, Fractals Everywhere, Academic Press, 1988
[4] Guojun Lu, Fractal image compression, Signal Processing: Image Communication, Volume 5, Issue 4, 1993
[5] Gupta, Shailender, Ankur Goyal, Bharat Bushan, "Information Hiding Using Least Significant Bit Steganography and Cryptography", I.J.Modern Education and Computer Science, 2012, 6, 27-34
[6] Vigier, Pierre, "Fractal Image Compression", https://pvigier.github.io/2018/05/14/fractal-image-compression.html, 2018, accessed 10th December 2023, 20.00 UTC+7.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Desember 2023

Edbert Eddyson Gunawan
13522039