Accredited Ranking SINTA 2
Decree of the Director General of Higher Education, Research and Technology, No. 158/E/KPT/2021
Validity period from Volume 5 Number 2 of 2021 to Volume 10 Number 1 of 2026

Published online on: **http://jurnal.iaii.or.id**

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 7 No. 1 (2023) 1292 - 1299 ISSN Media Electronic: 2580-0760

Digital Image Encryption Using Logistic Map

Muhammad Rizki[1], Erik Iman Heri Ujianto[2], Rianto[3]
[1,2,3]Magister Teknologi Informasi, Universitas Teknologi Yogyakarta, Yogyakarta, Indonesia
[1]6220211002.rizki@student.uty.ac.id, [2]erik.iman@uty.ac.id, [3]rianto@staff.uty.ac.id

Abstract

This study focuses on the application of the Logistic Map algorithm in Python programming language for Digital Image Encryption and Decryption, the differentiating aspect of the present research in contrast to prior studies lies in its elucidation of the practical application of the Logistic Map within the Python programming language, as opposed to the antecedent investigations which primarily confined their discourse to the realm of scientific and mathematical abstraction. Not only that, but it also investigates the impact of image type, image size, and Logistic Map parameter values on computational speed, memory usage, Encryption and Decryption results. Three image sizes (300px x 300px, 500px x 500px, and 1024px x 1024px) in TIFF, JPG, and PNG formats are considered. The Digital Image Encryption and Decryption process utilizes the Logistic Map algorithm implemented in Python. Various parameter values are tested for each image type and size to analyze the Encryption and Decryption outcomes. This research has effectively implemented the logistic map algorithm, resulting in the discovery of several significant findings. The findings indicate that image type does not affect memory usage, which remains consistent regardless of image type. However, image type significantly influences Decryption results and computation time. Notably, the TIFF image type exhibits the fastest computation time, with durations of 0.17188 seconds, 0.28125 seconds, and 1.10938 seconds for 300px x 300px, 500px x 500px, and 1024px x 1024px images, respectively. Additionally, the Decryption results vary depending on the image type. The Logistic Map algorithm is unable to restore Encryption results accurately for JPG images. Furthermore, this research highlights those higher values of x, Mu, and Chaos result in narrower histogram values, resulting in better encryption results, as evidenced by experiments using x=0.102, Mu=3.9 and Chaos=6400. This study contributes to the field by exploring the application of the Logistic Map algorithm in Python and analyzing the effects of image type, image size, and Logistic Map parameter values on computation time, memory usage, and Digital Image Encryption and Decryption results.

Keywords: logistic map; digital image encryption; python

1. Introduction

The development of technology forces many people to produce more multimedia data such as images, videos, audio, and so on. This development is also supported by the rapid dissemination of data on the internet [1]. Therefore, with the rapid advancement of communication technology, information exchange through the Internet is everywhere and has become a crucial part of many people's lives [2]. One type of data that is widely spread on the internet is images. An image contains a vast amount of information, not only in written form but also in descriptive visual form, making it essential to protect the privacy of the image owner.

Digital images or pictures, as a medium for storing information, are different from text data. Images have specific features such as size and values in each pixel [3]. One type of image that needs protection is medical images. According to research conducted by Kiran from Visvesvaraya Technological University (VTU) in 2020, numerous medical applications such as smart health, e-health, telemedicine, and others still use open networks to send and receive information related to medical images from both the client and server sides. These images contain highly sensitive information about patients' secrets. Unfortunately, these applications do not protect the images sent, mainly due to considering the time and computational burden involved in the process [4]. Not only in the medical field but also in other domains such as intellectual property and law, a higher level of security is necessary for image data [5].

There are many ways to protect data in the form of images, and one of them is by applying the Logistic Map theory. The Logistic Map is often used in various problem-solving scenarios, such as approximating the Specific Data Corona Virus [6], Enhance Chaos And Complexity Of Discrete Systems [7], Applying Dispersion Entropy To Status Characterization Of Rotary Machines [8], and Inverse Pheromone Approach

Accepted: 27-08-2023 | Received in revised: 22-10-2023 | Published: 26-11-2023

in a Chaotic Mobile Robot's Path Planning Based on a Modified Logistic Map [9]. This is because the Logistic Map is considered to have advantages in its simple functional structure with good chaos and controllable autocorrelation and cross-correlation properties [10]. Therefore, the Logistic Map may be also used in the process of protecting images.

Previous research has utilized the Logistic Map as an algorithm for encrypting images, as seen in a study conducted in 2019. In this research, a new approach to image encryption was proposed, based on a dual chaotic system using the two-dimensional Baker Chaotic Map to control the parameters and state variables of the Logistic Map. The goal was to extend the range and reduce the vulnerability level compared to the one-dimensional Logistic Map algorithm. The results of this research showed that the two-dimensional Baker Chaotic Map successfully increased the level of chaos to a point where it became unpredictable by complex analysis [11]. In the same year, another similar study discussed enhancing the chaotic behaviour of the Logistic Map. The research revealed that within the Logistic Map algorithm, a logistic sequence produces multiple random sequences that can be useful for controlling the random sequences of other logistic transformations. The study also analyzed key space, key sensitivity, correlation, information entropy, resistance to differential attacks, resilience analysis, and complexity analysis of the encrypted images. The results showed that the proposed algorithm could compete with other chaotic-based image encryption algorithms in terms of security and complexity [12]. Both studies demonstrate the potential of using the Logistic Map and its variations to create secure and chaotic image encryption algorithms, which are crucial in safeguarding sensitive image data from unauthorized access.

In the following year, there was research discussing the use of watermarks on images based on interconnected Logistic Maps and variants of optimization generated by Particle Swarm Optimization (PSO). The approach taken in this study involved decomposing the image using discrete wavelet transformation and Discrete Cosine Transform (DCT) techniques. Both transformations were applied to pixels with frequency levels that are insensitive to human perception, namely Low-High (LH) and High-Low (HL) pixels. The goal was to embed the watermark into the image in a way that the watermarked image still appeared like a regular image. The process of embedding the watermark into the image in this study involved applying the Logistic Map theory to chaotically destroy the watermark, making it difficult to read or recognize. The results of this research showed that the PSO algorithm used for multi-dimensional optimization of selecting DCT values efficiently found the optimal values for the strength of watermark embedding and DCT values [13]. Furthermore, there are several similar studies such as "An Efficient Image Encryption Scheme Based on S-Boxes and Fractional-Order Differential Logistic Map" [14], "An Image Encryption Scheme Based on Public Key Cryptosystem and Quantum Logistic Map" [15], "An Improved Digital Logistic Map and Its Application in Image Encryption" [16], and many other researches that utilize the Logistic Map as an algorithm for digital image encryption. All these studies demonstrate that the Logistic Map has proven to be an effective and reliable tool in protecting digital images, contributing to the development of security techniques in processing and exchanging images online.

However, most of these studies only discuss from a scientific perspective and the mathematical calculations alone. There has been no research that discusses in detail the technical implementation of the Logistic Map in a specific programming language, particularly in the Python programming language. This research aims to explain in detail the implementation of the Logistic Map as a digital image encryption algorithm using the Python programming language. Not only that, but this research also analyzes the influence of each parameter change possessed by the Logistic Map on encryption results, execution time, and memory usage during the computation process.

2. Research Methods

This research followed the steps shown in Figure 1 as a method for conducting experiments and analysis.

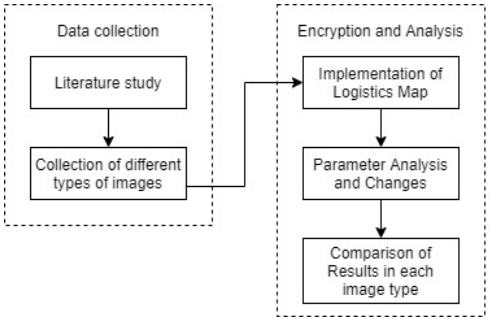

Figure 1. Flow of Research Methods

2.1 Data Collection

The data collection process involves several stages, including a literature review phase aimed at gathering various references. The literature review is also useful for collecting scientific evidence related to the differences between previous research and this study. The data collection phase also includes gathering various types of images such as JPG, PNG, and TIFF. Each image comes in three-pixel sizes: 300px x 300px, 500px x 500px, and 1024px x 1024px.

2.2 Image Encryption

The Encryption and analysis process includes the following steps:

DOI: https://doi.org/10.29207/resti.v7i6.5389

This research utilizes the Logistic Map algorithm for encrypting digital images in the form of pictures. The Logistic Map is an algorithm used to introduce chaos or randomness, as described by Formula 1 [17].

$$Xi + 1 = \mu Xi(1 - Xi) \quad (1)$$

The behaviour of the Logistic Map can be highly complex and exhibit various patterns depending on the value of μ (Mu) that is specified. For example, if the value of μ lies between 0 and 1, the population will decrease over time and eventually reach 0. However, if the value of μ lies between 1 and 3, the population will oscillate between two values, and if μ lies between 3 and 3.57, the population will oscillate between four values. When the value of μ is between 3.57 and 3.57·(3.57-3), the function will exhibit period-doubling bifurcation, meaning each population will be isolated between 2^i values as the value of i increases. This behaviour will continue until the value of μ reaches 4.6692, at which point the population will exhibit chaos.

Here is the example of chaos generated by the Logistic Map with an initial value of X(0) = 0.5 and μ = 3.8 for a total of 12 iterations:

$x(0) = 0.95$

$x(1) = \mu * 0.5 * (1 - 0.5 = 0.95$

$x(2) = \mu * 0.95 * (1 - 0.95) = 0.1805$

$x(3) = \mu * 0.1805 * (1 - 0.1805) = 0.59732$

$x(4) = \mu * 0.59732 * (1 - 0.59732) = 1.13707$

$x(5) = \mu * 1.13707 * (1 - 1.13707) = -1.99852$

$x(6) = 3.8 * (-1.99852) * (1 - (-1.99852)) = -14.4873$

$x(7) = 3.8 * (-14.4873) * (1 - (-14.4873)) = -268.486$

$x(8) = 3.8 * (-268.486) * (1 - (-268.486)) = -183412.9$

$x(9) = 3.8 * (-183412.9) * (1 - (-183412.9)) = -124023264$

$x(10) = 3.8 * (-124023264) * (1 - (-124023264)) = -99329897485.2$

$x(11) = 3.8 * (-99329897485.2) * (1 - (-99329897485.2)) = -9.93297E + 19$

$x(12) = 3.8 * (-9.93297E + 19) * (1 - (-9.93297E + 19)) = -Infinity$

This stage aims to analyze the influence of parameter changes in the Logistic Map on the encrypted image results. Subsequently, the encryption results for each type of image will be compared based on the execution time for encryption and decryption, as well as the memory usage during each process.

3. Results and Discussions

After conducting a literature review and data collection, the next step is to implement the Logistic Map using the Python programming language. This stage utilizes the Pillow library (pillow.readthedocs.io) as a module for image processing and Numpy (numpy.org) as a module for mathematical function engineering, which will be used to compute Formula 1.

3.1 Application of Digital Image Encryption

The stages of image encryption using Logistic Map in the Python programming language can be displayed in Figure 2.

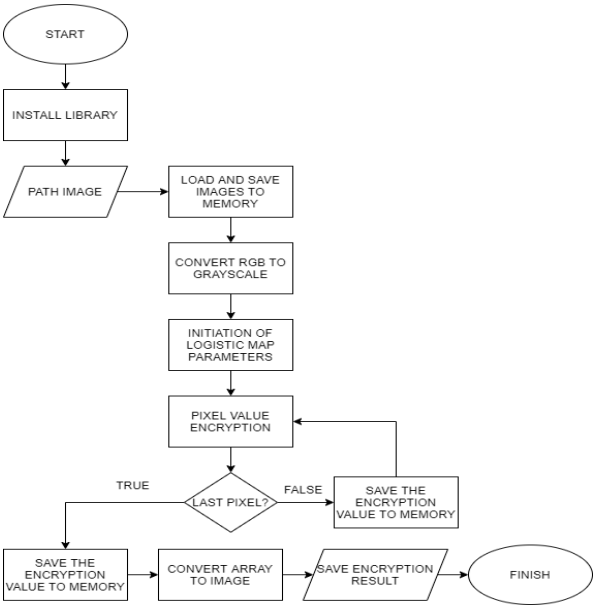

Figure 2. Encryption Flow

Install Library is the first stage to perform image encryption in this research is to load the Pillow and Numpy libraries, as shown in Figure 3.

```
# Load all libraries
from PIL import Image
import numpy as np
```

Figure 3. Install Library

Input Path Image is the stage where you need to select the image that will be encrypted using the code indicated in Figure 4.

```
# initialize path image
img_file = 'image.png'
path_real_img = 'image/image_real/'+ img_file
path_enc = 'image/enc/'+ img_file
path_dec = 'image/dec/'+ img_file
```

Figure 4. Path Image

DOI: https://doi.org/10.29207/resti.v7i6.5389

The code shown in Figure 4 aims to fetch the file "image.png" and initialize the path of the folder where the encrypted and decrypted results will be stored. The folder path used in this research is shown in Figure 5.

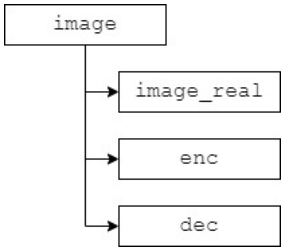

Figure 5. Structure folder

The image selected as the test image at this stage is an image with a size of 300px x 300px as shown in Figure 6.

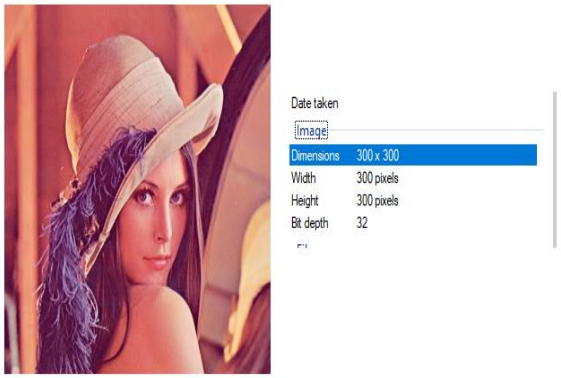

Figure 6. Trial image size

Loading and saving an image to memory stage is before the image can be processed, the image must first be saved into memory, the following Python code to save the image to memory is shown in Figure 7.

```
# load image
img_real = Image.open(path_real_img)
display(img_real)
```

Figure 7. Save image to memory (variable) img_real

The line of code shown in Figure 7 also aims to display the original image before the encryption process is run.

The stage of converting images from RGB (Red, Green, Blue) format to Grayscale format aims to reduce the amount of memory and the size of the image, thus, the calculation process will be faster. Python can change the image format from RGB to Grayscale in several ways, one of which is by utilizing the Pillow library as shown in Figure 8.

```
# Convert RGB to Gray Scale
im = Image.open(path_real_img).convert('L')
display(im)
# Convert image to array format
im_array = np.array(im)
print(im_array)
```

Figure 8. Converting RGB Format to Grayscale

The code displayed in Figure 8 above produces new data, where the previous data has a 3-dimensional form with a size of 300 x 300 x 4 into 2-dimensional data with a size of 300 x 300 as displayed in Figure 9.

RGB Format

```
[[[231 134 135 255]
  [234 140 135 255]
  [238 143 143 255]
  ...
  [ 90  13  65 255]
  [ 89  13  58 255]
  [ 80  14  56 255]]

 [[232 128 127 255]
  [233 140 133 255]
  [238 149 137 255]
  ...
```

Gray Scale

```
[[163 168 171 ...  42  41  39]
 [159 167 174 ...  46  42  37]
 [159 163 173 ...  43  48  40]
 ...
 [157 171 183 ...  72  55  54]
 [156 167 180 ...  55  52  53]
 [156 166 177 ...  51  47  57]]
```

Figure 9. RGB to Grayscale format result

The stages of parameter initialization for the Logistic Map can be seen in Figure 10.

```
# Initialized Chaostic Parameters
keystream = []
x = 0.002
Mu = 4
chaos = 3064
```

Figure 10. Parameter Initialization

The purpose of initializing parameters in the Logistic Map is to determine the level of chaos generated to be applied to each pixel that will be encrypted.

Encrypting the values in each pixel stage is a crucial step, where the encryption process involves the utilization of two functions. The first function is used to define the Logistic Map, as illustrated in Figure 11.

```
# Generate a keystream using a
chaotic map (e.g. logistic map)

def logistic_map(x, r):
    return r * x * (1 - x)
```

Figure 11.Implementation of the Logistic Map

The second function serves as the engine for encrypting each pixel, as demonstrated in Figure 12.

DOI: https://doi.org/10.29207/resti.v7i6.5389

```
# Encrypted Machine

for i in range(im_array.size):
    x = logistic_map(x, Mu)
    _x = int(x * chaos)
    keystream.append(_x)
    print("generate pixel {} value
{}".format(i, _x))
```

Figure 12.Encryption Engine

The functions shown in Figures 11 and 12 are interconnected, as the function depicted in Figure 11 is the implementation of the Logistic Map in the Python programming language. This function is responsible for introducing chaos to each pixel value provided by the function depicted in Figure 12. Here is a simulation of the encryption calculation performed by these two functions.

$$x1 = 4 * 0.002 * (1 - 0.002) = 0.007984$$

$$x2 = 4 * 0.007984 * (1 - 0.007984) = 0.031681022976$$

$$x3 = * 0.031681022976 * (1 - 0.031681022976) = 0.12270934303677665$$

……..

$$x90000 = 4 * 0.917356585505951 * (1 - 0.917356585505951) = 0.3032539221392552$$

Due to the image dimensions being processed being 300 x 300, the calculations will continue until the last pixel, which is the 90,000th pixel. Not only that, but the encryption engine shown in above Figure 12 also handles the calculations and rounding of the multiplication results between the Logistic Map values and chaos values. Here is a simulation of the calculations :

$$x1 = 0.007984 * 3064 = 24.462975999999998 = 24$$

$$x2 = 0.031681022976 * 3064 = 97.070654398464 = 97$$

$$x3 = 0.12270934303677665 * 3064 = 375.98142706468366 = 375$$

……..

$$x90000 = 0.3032539221392552 * 3064 = 929.170017434678 = 929$$

The multiplication result of the Logistic Map with the chaos value above is the final value of this stage.

Save the encryption values to memory as shown in Figure 12 is not limited to just the function for encrypting each pixel value contained within the test image. The function depicted in Figure 12 is also used to store the encrypted data results. This function is executed by adding each encrypted value to a variable initialized with the name 'keystream'. The encrypted results stored in this variable can be seen in Figure 13.

```
array([  24,   97,  375, ..., 1972, 2810,  929])
```

Figure 13.Value of the 'keystream' Variable

The Bitwise XOR operation is used to scramble the previously encrypted values, and the result of this operation can be seen in Figure 14.

```
[[ 186  195  470 ... 2611 1139 2830]
 [ 266 1300 2896 ...  768 2692 1692]
 [2901  162  173 ...  959 2481 2087]
 ...
 [  17  162  540 ... 1601 2966   89]
 [  63  107  341 ...  633 1672 3016]
 [ 262 1034 2755 ... 2012 2716  973]]
```

Figure 14. Value of the 'keystream' Variable

Converting an Array to an Image is done after the Bitwise XOR process is completed. The result of the digital image encryption process can be observed in Figure 15.

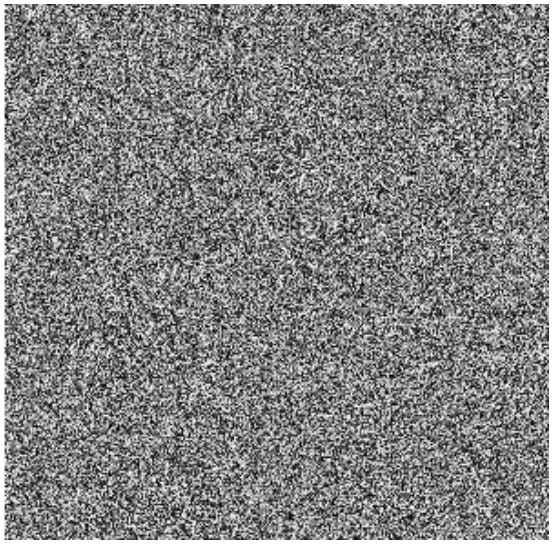

Figure 15. Encryption Result

After the encryption is successfully performed, the next step is to save the image into the previously prepared folder, as indicated in Figure 5.

3.1 Analysis of Parameter Changes in Encryption Results, Memory Usage, and Computational Speed

After implementing the Logistic Map using the Python programming language, the next step is to analyze the parameter changes in encryption results, memory usage, and computational speed for each type and size of the image. The test results can be seen in Table 1.

DOI: https://doi.org/10.29207/resti.v7i6.5389

Table 1. Experiment Results

No	Type	Pixel	x	Mu	Chaos	Encryption Size (Kb)	Encryption Speed (s)	Decryption Size (Kb)	Decryption Speed (s)
1	JPG	300x300	0.002	3	200	351.67188	0.21875	351.68750	0.03125
2	JPG	500 x 500	0.002	3	200	976.67188	0.50000	976.68750	0.06250
3	JPG	1024x1024	0.002	3	200	4096.10938	1.54688	4096.12500	0.07813
4	PNG	300x300	0.002	3	200	351.67188	0.28125	351.68750	0.03125
5	PNG	500x500	0.002	3	200	976.67188	0.43750	976.68750	0.12500
6	PNG	1024x1024	0.002	3	200	4096.10938	2.00000	4096.12500	0.51563
7	TIFF	300x300	0.002	3	200	351.67188	0.17188	351.68750	0.01563
8	TIFF	500x500	0.002	3	200	976.67188	0.29688	976.68750	0.04688
9	TIFF	1024x1024	0.002	3	200	4096.10938	1.26563	4096.12500	0.04688
10	JPG	300x300	0.032	3.5	1200	351.67188	0.20313	351.68750	0.01563
11	JPG	500x500	0.032	3.5	1200	976.67188	0.32813	976.68750	0.03125
12	JPG	1024x1024	0.032	3.5	1200	4096.10938	1.25000	4096.12500	0.03125
13	PNG	300x300	0.032	3.5	1200	351.67188	0.23438	351.68750	0.03125
14	PNG	500x500	0.032	3.5	1200	976.67188	0.40625	976.68750	0.04688
15	PNG	1024x1024	0.032	3.5	1200	4096.10938	1.70313	4096.12500	0.42188
16	TIFF	300x300	0.032	3.5	1200	351.67188	0.18750	351.68750	0.01563
17	TIFF	500x500	0.032	3.5	1200	976.67188	0.39063	976.68750	0.03125
18	TIFF	1024x1024	0.032	3.5	1200	4096.10938	1.23438	4096.12500	0.04688
19	JPG	300x300	0.102	3.9	6400	351.67188	0.20313	351.68750	0.01563
20	JPG	500x500	0.102	3.9	6400	976.67188	0.34375	976.68750	0.03125
21	JPG	1024x1024	0.102	3.9	6400	4096.10938	1.28125	4096.12500	0.04688
22	PNG	300x300	0.102	3.9	6400	351.67188	0.20313	351.68750	0.03125
23	PNG	500x500	0.102	3.9	6400	976.67188	0.34375	976.68750	0.04688
24	PNG	1024x1024	0.102	3.9	6400	4096.10938	1.73438	4096.12500	0.50000
25	TIFF	300x300	0.102	3.9	6400	351.67188	0.17188	351.68750	0.01563
26	TIFF	500x500	0.102	3.9	6400	976.67188	0.28125	976.68750	0.03125
27	TIFF	1024x1024	0.102	3.9	6400	4096.10938	1.10938	4096.12500	0.03125

The testing was conducted using the parameters x = 0.002, Mu = 3, and $Chaos$ = 200 for the first test, x = 0.032, Mu = 3.5, and $Chaos$ = 1200 for the second test, and x = 0.102, Mu = 3.9, and $Chaos$ = 6400 for the subsequent test.

Memory Usage Analysis: based on the conducted test results, it was found that the image type and Logistic Map parameter values have no significant impact and remain consistent on the memory usage during the computation process. The memory usage for images of size 1024px x 1024px is 4096.12500kb, while for size 500px x 500px it is 976.68750kb, and for size 300px x 300px, it is 351.68750kb. The memory usage difference during decryption for each size is 0.00038146% for 1024px x 1024px size, 0.001599821% for 500px x 500px size, and 0.004443062% for 300px x 300px size.

Computational Speed Analysis: the conducted experiments have demonstrated that the TIFF image format proves to be the fastest in terms of computational performance compared to JPG and PNG. While JPG and PNG formats take 0.20313 seconds for a 300px x 300px image, for images of sizes 500px x 500px and 1024px x 1024px, the JPG format can outperform the PNG format, although it still lags the computational speed achieved by the TIFF format.

The Logistic Map algorithm can perform encryption on TIFF images in 0.17188 seconds for a 300px x 300px image, 0.28125 seconds for a 500px x 500px image, and 1.10938 seconds for a 1024px x 1024px image. Moreover, the test results indicate that the primary factors affecting encryption speed are the image type and size, while the Logistic Map parameter values have a relatively minor influence on encryption speed.

Encryption Result Analysis: the conducted tests for each image size and type demonstrate that the image type and size do not influence the encryption results. The primary factor affecting encryption results is the parameter values of the Logistic Map, as illustrated in Figure 16.

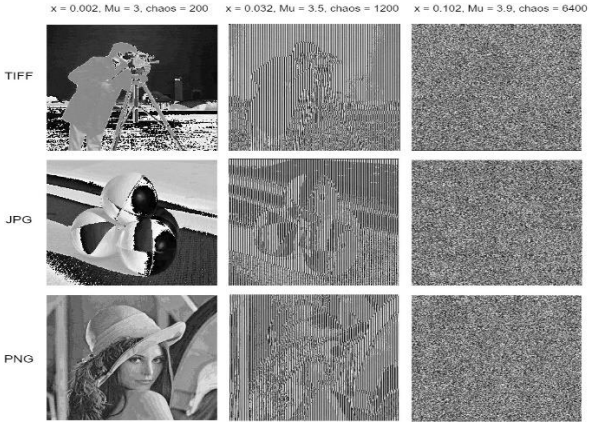

Figure 16. Encryption Results

The experiment shown in Figure 16 indicates that higher values of x, Mu, and Chaos lead to more randomized encryption of pixel values. Moreover, the values of x, Mu, and Chaos significantly influence the

DOI: https://doi.org/10.29207/resti.v7i6.5389

encryption patterns generated by the Logistic Map, as illustrated in Figure 17.

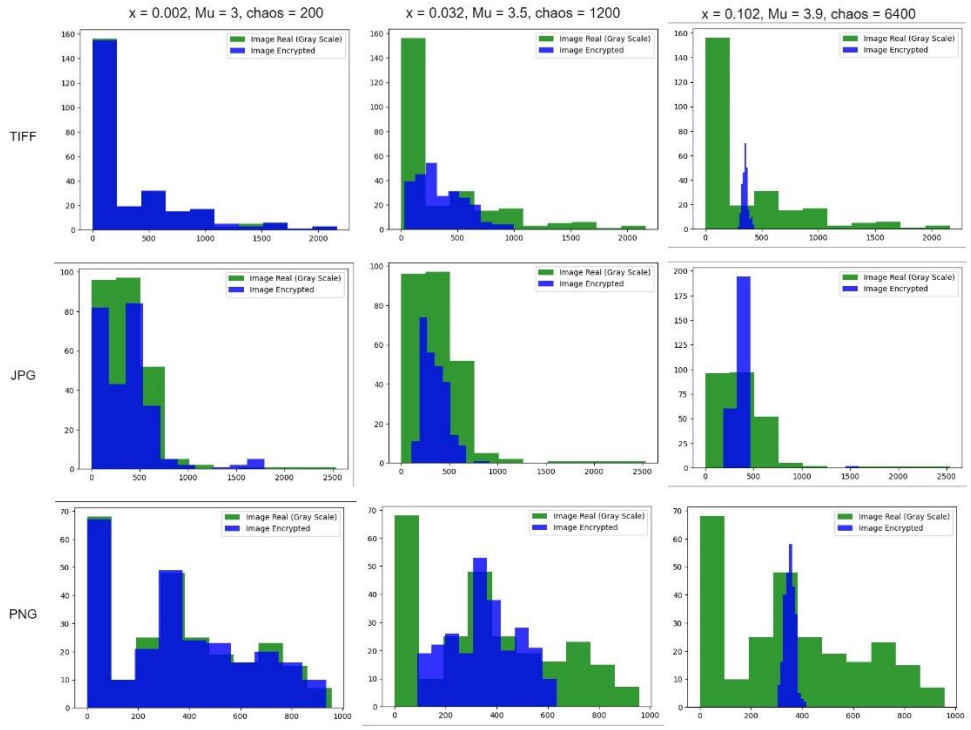

Figure 17. The Influence of Logistic Map Parameter Values on Encryption Patterns

Based on Figure 17, when the values are x = 0.002, Mu = 3, and Chaos = 200, the histogram values generated by the encrypted image closely resemble the histogram values of the original image. This results in the encrypted image still being recognizable. Moving forward, when the Logistic Map parameters are set to x = 0.032, Mu = 3.5, and Chaos = 1200, the histogram range of the encrypted image becomes narrower than before. This prompts the Logistic Map to produce more randomized encryption, yet the encryption quality is still insufficient as the encrypted result remains somewhat discernible, as shown in Figure 16. Furthermore, when the parameters are x = 0.102, Mu = 3.9, and Chaos = 6400, the pixel range on the histogram graph drops below 500 pixels for each image type tested. This chaotic behaviour affects every pixel, rendering the encrypted image almost unrecognizable.

Encryption Result Analysis: this research has found that one of the factors influencing the decryption results performed by the Logistic Map is the image type, as illustrated in Figure 18.

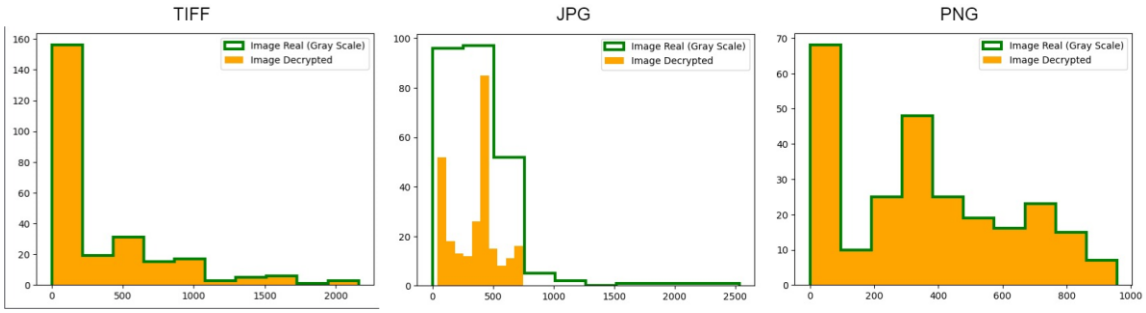

Figure 18. The Influence of Image Type on Decryption Results

Figure 18 illustrates that TIFF and PNG image types can restore the encryption values to their original state, enabling the decryption results to match the histogram values of the original image. On the other hand, the decryption results produced by the JPG image type still do not fully restore the original values, resulting in some noise or artefacts in the decrypted image, as shown in Figure 19.

DOI: https://doi.org/10.29207/resti.v7i6.5389

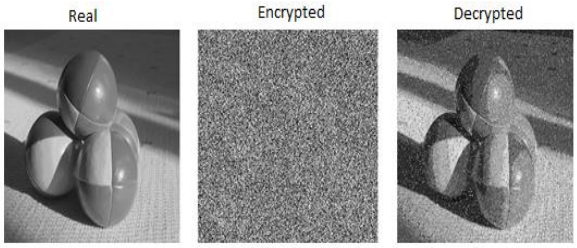

Figure 19. Comparison between the Original Image and the Decrypted Image in JPG Image Type

Although the decryption results still exhibit some noise or artefacts in the JPG image type, the objects in the decrypted image are still recognizable. The decryption result shown in Figure 19 is obtained from the encrypted image generated by the Logistic Map with parameter values x = 0.102, Mu = 3.9, and Chaos = 6400. This value is also applicable to the histogram graph shown in Figure 18. Based on the previous test results, this parameter combination is chosen as it produces the best encryption result.

4. Conclusion

This research utilized the Python programming language to perform image encryption using the Logistic Map algorithm. The Python programming language incorporates libraries such as Numpy and Pillow to facilitate image computation and complex mathematical operations. The results of the conducted study demonstrate that the Logistic Map can execute encryption and decryption rapidly, yielding excellent outcomes for TIFF image types. Moreover, the JPG image type also exhibits computational speeds surpassing those of PNG image types. However, the JPG image type still introduces noise in the decryption results produced by the Logistic Map. The amount of memory used during the computational process remains consistent for each image type, with image size being one of the factors influencing memory usage. On the other hand, parameters such as x, Mu, and Chaos only affect the outcomes and encryption patterns generated by the Logistic Map algorithm. This research converts RGB images to grayscale, aiming to streamline the calculations performed by the Logistic Map, thus optimizing memory usage and computational efficiency. However, this conversion results in the decrypted images remaining in grayscale format. Therefore, future studies are encouraged to explore similar research that generates decrypted images in RGB format.

References

[1] V. Kumar and A. Girdhar, "A 2D logistic map and Lorenz-Rossler chaotic system based RGB image encryption approach," *Multimed. Tools Appl.*, vol. 80, no. 3, pp. 3749–3773, Jan. 2021, doi: 10.1007/s11042-020-09854-x.

[2] X. Liu, D. Xiao, and C. Liu, "Three-level quantum image encryption based on Arnold transform and logistic map," *Quantum Inf. Process.*, vol. 20, no. 1, Jan. 2021, doi: 10.1007/s11128-020-02952-7.

[3] M. Muñoz-Guillermo, "Image encryption using a q-deformed logistic map," *Inf. Sci. (Ny).*, vol. 552, pp. 352–364, Apr. 2021, doi: 10.1016/j.ins.2020.11.045.

[4] Kiran, B. D. Parameshachari, H. T. Panduranga, and S. L. Ullo, "Analysis and Computation of Encryption Technique to Enhance Security of Medical Images," in *IOP Conference Series: Materials Science and Engineering*, Oct. 2020, vol. 925, no. 1. doi: 10.1088/1757-899X/925/1/012028.

[5] M. Sharma, "Image encryption based on a new 2D logistic adjusted logistic map," *Multimed. Tools Appl.*, vol. 79, no. 1–2, pp. 355–374, Jan. 2020, doi: 10.1007/s11042-019-08079-x.

[6] N. Kyurkchiev, A. Iliev, and A. Rahnev, "ON THE HALF–LOGISTIC MODEL WITH "POLYNOMIALVARIABLE TRANSFER". APPLICATION TO APPROXIMATETHE SPECIFIC "DATA CORONA VIRUS"," *Int. J. Differ. Equations Appl.*, vol. 19, no. 1, pp. 45–6103, 2020, doi: 10.12732/ijdea.v19i1.4.

[7] H. Natiq, S. Banerjee, and M. R. M. Said, "Cosine quantification technique to enhance chaos and complexity of discrete systems," *Eur. Phys. J. Spec. Top.*, vol. 228, no. 1, pp. 185–194, May 2019, doi: 10.1140/epjst/e2019-800206-9.

[8] M. Rostaghi, M. R. Ashory, and H. Azami, "Application of dispersion entropy to the status characterization of rotary machines," *J. Sound Vib.*, vol. 438, pp. 291–308, Jan. 2019, doi 10.1016/j.jsv.2018.08.025.

[9] E. K. Petavratzis *et al.*, "An Inverse Pheromone Approach in a Chaotic Mobile Robot's Path Planning Based on a Modified Logistic Map," *Technologies*, vol. 7, no. 4, p. 84, Dec. 2019, doi: 10.3390/technologies7040084.

[10] G. Ye and X. Huang, "An efficient symmetric image encryption algorithm based on an intertwining logistic map," *Neurocomputing*, vol. 251, pp. 45–53, Aug. 2017, doi: 10.1016/j.neucom.2017.04.016.

[11] Y. Luo, J. Yu, W. Lai, and L. Liu, "A novel chaotic image encryption algorithm based on improved baker map and logistic map," *Multimed. Tools Appl.*, vol. 78, no. 15, pp. 22023–22043, Aug. 2019, doi: 10.1007/s11042-019-7453-3.

[12] R. Li, Q. Liu, and L. Liu, "Novel image encryption algorithm based on improved logistic map," *IET Image Process.*, vol. 13, no. 1, pp. 125–134, 2019, doi: 10.1049/iet-ipr.2018.5900.

[13] X. Kang, Y. Chen, F. Zhao, and G. Lin, "Multi-dimensional particle swarm optimization for robust blind image watermarking using intertwining logistic map and hybrid domain," *Soft Comput.*, vol. 24, no. 14, pp. 10561–10584, Jul. 2020, doi: 10.1007/s00500-019-04563-6.

[14] Y. Q. Zhang, J. L. Hao, and X. Y. Wang, "An Efficient Image Encryption Scheme Based on S-Boxes and Fractional-Order Differential Logistic Map," *IEEE Access*, vol. 8, pp. 54175–54188, 2020, doi: 10.1109/ACCESS.2020.2979827.

[15] G. Ye, K. Jiao, X. Huang, B. M. Goi, and W. S. Yap, "An image encryption scheme based on public key cryptosystem and quantum logistic map," *Sci. Rep.*, vol. 10, no. 1, Dec. 2020, doi: 10.1038/s41598-020-78127-2.

[16] H. Xiang and L. Liu, "An improved digital logistic map and its application in image encryption," *Multimed. Tools Appl.*, vol. 79, no. 41–42, pp. 30329–30355, Nov. 2020, doi: 10.1007/s11042-020-09595-x.

[17] Iqbal, Kusrini, and A. Nasiri, "KOMPARASI HASIL ENKRIPSI ARNOLD CAT MAP DAN LOGISTIC MAPPADA CITRA DIGITAL," *J. Ilm. Inf. Technol. d'Computare* , vol. 10, pp. 10–16, Jun. 2020.

DOI: https://doi.org/10.29207/resti.v7i6.5389