applied sciences

MDPI

Article

Image-to-Image Steganography with Josephus Permutation and Least Significant Bit (LSB) 3-3-2 Embedding

Muhammad Rafly Yanuar [1], Suryadi MT [2], Catur Apriono [1,*] and Muhammad Firdaus Syawaludin [1,*]

[1] Department of Electrical Engineering, Universitas Indonesia, Depok 16424, Indonesia; raflyyyanuar@gmail.com
[2] Department of Mathematics, Universitas Indonesia, Depok 16424, Indonesia; yadi.mt@sci.ui.ac.id
* Correspondence: catur@eng.ui.ac.id (C.A.); m.firdaus04@ui.ac.id (M.F.S.)

Abstract: In digital image security, the Josephus permutation is widely used in cryptography to enhance randomness. However, its application in steganography is underexplored. This study introduces a novel method integrating the Josephus permutation into the LSB 3-3-2 embedding technique for image steganography. This approach improves the randomness of the keystream generated by the chaotic logistic map, addressing vulnerabilities in basic logistic maps susceptible to steganalysis. Our algorithm is tested on RGB images as secret data, presenting higher complexity compared to grayscale images used in previous studies. Comparative analysis shows that the proposed algorithm offers higher payload capacity while maintaining image quality, outperforming traditional LSB techniques. This research advances the field of image steganography by demonstrating the effectiveness of the Josephus permutation in creating more secure and robust steganographic images.

Keywords: Josephus permutation; image steganography; LSB 3-3-2

check for updates

Citation: Yanuar, M.R.; MT, S.; Apriono, C.; Syawaludin, M.F. Image-to-Image Steganography with Josephus Permutation and Least Significant Bit (LSB) 3-3-2 Embedding. *Appl. Sci.* **2024**, *14*, 7119. https://doi.org/10.3390/app14167119

Academic Editor: David Megías

Received: 20 June 2024
Revised: 26 July 2024
Accepted: 8 August 2024
Published: 14 August 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Steganography is one of the crucial fields in information security that aims to conceal sensitive and secret data within seemingly ordinary media [1]. Unlike cryptography, which transforms the appearance of secret images into suspicious ones, steganography does not arouse suspicion from parties observing the resulting images. Steganography achieves this by hiding secret images within another image (i.e., cover image), so its existence is concealed behind the cover image. This additional layer of security is an effort of "security through obscurity", an advantage that steganography has over cryptography [2].

Despite the existing advancements in steganography, there is a continuous need for enhancing the security and robustness of these methods, especially in the context of digital images. One area that has shown promise in this regard is the use of permutation techniques to randomize the embedding locations of secret data [3,4]. The Josephus permutation, traditionally used in cryptographic applications [5,6], offers a novel approach to enhancing the randomness and security of steganographic methods.

In this study, we propose a novel steganography algorithm that integrates the Josephus permutation and a 3-3-2 LSB embedding scheme. The Josephus permutation is employed to enhance the randomness of pixel distribution locations derived from the keystream generated by a chaotic logistic map. Subsequently, the 3-3-2 LSB embedding technique is utilized to embed a secret RGB image into a cover RGB image. The incorporation of the Josephus permutation into the keystream serves to extend the security obscurity period of the LSB 3-3-2 embedding.

Our approach explores the use of the Josephus permutation in image steganography, which has not been extensively studied before. Historically, the Josephus permutation has been applied primarily in cryptographic scenarios. By introducing this technique into the steganographic process, we aim to leverage its ability to generate highly random and non-repetitive sequences, thereby enhancing the security of the embedded data.

As far as our knowledge, unlike existing methods which often use LSB techniques for embedding text or grayscale images, our approach is also the first to employ the LSB 3-3-2 method for RGB-to-RGB image steganography, representing a new way of using the LSB 3-3-2 in the context of image steganography. Prior research has primarily focused on grayscale images, which do not fully utilize the potential of color information in enhancing security and capacity.

By addressing these gaps, our proposed method aims to advance the field of image steganography by providing a more secure and efficient technique for hiding color images. This research offers practical implications for its application in secure communications.

2. Literature Review

The field of image steganography has seen significant advancements in recent years, with researchers exploring various techniques to enhance the security, capacity, and imperceptibility of hidden data. This literature review delves into the recent progress in image steganography, highlighting key developments and methodologies. Following this, the review examines the application of the Josephus permutation in cryptography, demonstrating its effectiveness in improving randomness and security. Finally, the review explores the role of chaos functions in image steganography, emphasizing their unique properties that contribute to the robustness and efficiency of steganographic algorithms.

2.1. Image Steganography

The implementation of image steganography encompasses several variations, primarily based on the embedding techniques used. There are several embedding methods that can be used in the context of image steganography, including statistical, machine learning (ML), deep learning (DL), and quantum methods [7]. Among these, statistical methods in the form of Least Significant Bit (LSB) embedding is the simplest and most flexible [8].

Despite advancements in other techniques, the fundamental LSB techniques still hold significant relevance. Recent studies, such as by Patra and De [9], demonstrated the continued importance and effectiveness of LSB-based methods in scenarios where simplicity and computational efficiency are crucial. Elharrouss et al. [10] proposed a novel approach by combining the *k*-LSB method with a pseudo-random number generator to select pixels for embedding. In this scheme, the variable *k* represents the number of bits from the cover image embedded in the secret image. This technique improves the level of security by randomizing the embedding positions. Astuti's research [11] demonstrated the embedding of a grayscale secret image into an RGB cover image using LSB-1, LSB-2, and LSB-4 techniques. Their findings indicated that LSB-1 and LSB-2 exhibit small payload sizes, while LSB-4 result in poor Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) values.

Previous studies have primarily focused on embedding text or grayscale images using various LSB schemes [11]. However, the use of basic logistic maps in LSB techniques has shown vulnerabilities to steganalysis due to their predictable patterns [12]. This highlights the need for increased randomness to ensure better security. In this study, we employ the Josephus permutation to significantly enhance the randomness of the embedding locations, thereby mitigating the risk of detection by steganalysis.

Our inspiration for using the Josephus permutation comes from its successful application in cryptography to increase randomness. The Josephus permutation has demonstrated its effectiveness in improving scrambling effects and permutation efficiency, making it a valuable tool in cryptographic algorithms [13]. By leveraging this technique in steganography, we aim to enhance the robustness and security of our method, ensuring that the embedded data remain well protected against various types of attacks.

Further, our method embeds RGB images as secret data, which presents a higher level of complexity compared to grayscale images used in previous works [11]. This not only extends the work of [11] by exploring the use of RGB images but also serves as a rigorous test of the effectiveness of the Josephus permutation in handling more complex

data. The use of RGB images allows for a more detailed and nuanced representation of the secret data, which, when combined with the enhanced randomness from the Josephus permutation, provides a more secure and robust steganographic technique.

2.2. Josephus Permutation for Image Cryptography

The Josephus permutation is a classical method of rearranging sequences named after Flavius Josephus, a Jewish–Roman historian [14]. Originating from his historical account "The Jewish War", the method recounts an event where Josephus and 40 soldiers faced capture in a cave, opting for a structured mass suicide in a circle. Each soldier killed the one (i.e., $k = 1$) to their left until one remained. Unexpectedly, Josephus strategically positioned himself to be the last survivor. This Josephus permutation is depicted in Figure 1.

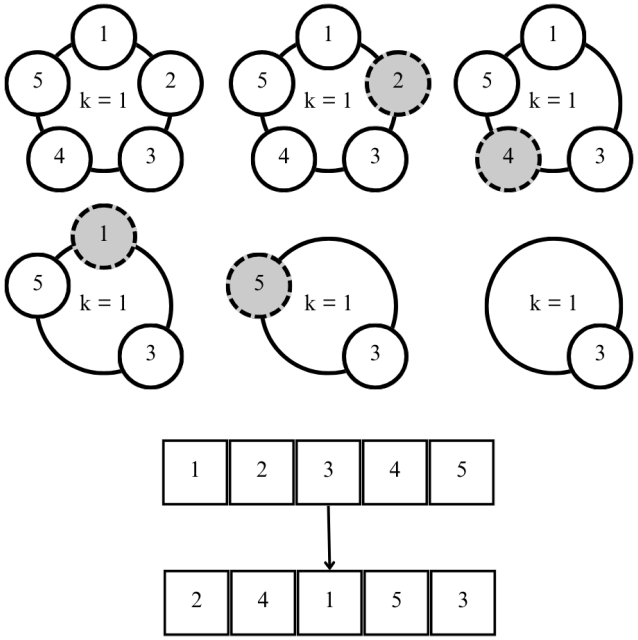


Figure 1. Josephus permutation of the number sequence 1, 2, 3, 4, 5 with a fixed key of one produces a new sequence 2, 4, 1, 5, 3.

Table 1 shows where someone must stand to survive among n people. In Josephus's case with 40 soldiers, he had to stand in the 19th position. The survivor's position can also be efficiently calculated using the binary representation, where the number of soldiers n undergoes the "Rotate Left" operation, i.e., the leftmost bit position (Most Significant Bit or MSB) becomes the rightmost bit position (Least Significant Bit or LSB) in its binary order. This new binary arrangement is the position where someone can survive. It is important to know that the standard Josephus permutation uses a constant key of $k = 1$.

Table 1. Number of soldiers n and the survivor's position $S(n)$.

n	$S(n)$
1	1
41	19
100	73

However, the Josephus permutation-based cryptography methods do not focus on who ends up at the last turn but rather on the execution sequence. In other words, for a sequence of numbers totaling n, the order from the first to "dead" and to the "survivor" will be calculated. Wang et al. [15], Hua et al. [16], and Liang et al. [17] used this type of implementation to encrypt secret images and achieve confusion in their research, making

the encrypted images unrecognizable compared to the original ones. Research conducted by Setiadi et al. [18] demonstrated the capability of the Josephus permutation in modifying bit levels for cryptography. This modification, which is easy to perform, offers a greater possibility of generating different bit combinations from the original bits. Consequently, the changes in pixels become more significant.

It is evident that the Josephus permutation is more saturated in data security in cryptography and less utilized in data security in steganography. Therefore, this study opts for combining the Josephus permutation and LSB embedding techniques, specifically, LSB 3-3-2, for image steganography.

2.3. Chaos Function

The application of chaos functions in LSB steganography continues to be relevant due to its simplicity and effectiveness in generating pseudo-random sequences [19]. A function is considered chaotic if it demonstrates sensitive dependence on the initial conditions, is topologically transitive, and has periodic points that are dense in its domain [20].

Sensitive dependence on the initial conditions means that small differences in the initial conditions lead to significantly divergent outcomes over time [21,22]. If two slightly different initial values lead to subsequent values that increasingly diverge over time, this condition is called chaotic. The rate at which these two values diverge from each other over a given period is represented by a value known as the Lyapunov exponent [23].

In addition to sensitive dependence, a chaotic function must also be random or topologically transitive. This characteristic can be illustrated using a bifurcation diagram, which visually represents the qualitative changes in the dynamics of a system as control parameters are varied. According to Kocarev [24], bifurcation is defined as a qualitative change in the dynamics of a dynamical system given a change in the control parameters. Thus, a bifurcation diagram is a visual representation of these changes in the behavior of a dynamical system as its control parameters are altered.

Several chaos functions can be used to generate key sequences for steganography, including the logistic map, tent map, quadratic map, Bernoulli map, sine map, and Chebyshev map [25]. Each of these functions has specific intervals where they exhibit chaotic behavior, which can be determined using methods like the Lyapunov exponent and bifurcation diagrams. This research focuses on the logistic map due to its well-documented chaotic properties and effectiveness in steganographic applications [25]. It was initially used to model population dynamics but was later found to exhibit chaotic behavior [26]. The map is represented by the equation:

$$x_{n+1} = r \cdot x_n(1 - x_n) \tag{1}$$

where x_n represents the population size at time n, and r is the growth parameter.

3. Methodologies

The process of embedding and extracting the secret image to and from the cover image and stego image, respectively, is shown in Figure 2. The first step involves using the logistic map to generate a keystream. The effectiveness of the logistic map in generating this keystream is assessed using the Lyapunov exponent and bifurcation diagram, which help determine the parameter ranges that induce chaos. Once these parameters are established, a new keystream is generated. The generated keystream is then scrambled using the Josephus permutation to ensure efficiency and complexity. Finally, the scrambled keystream is used to determine the embedding locations for the secret image within the cover image using the LSB 3-3-2 technique. This technique is also employed to extract the secret image from the cover image.

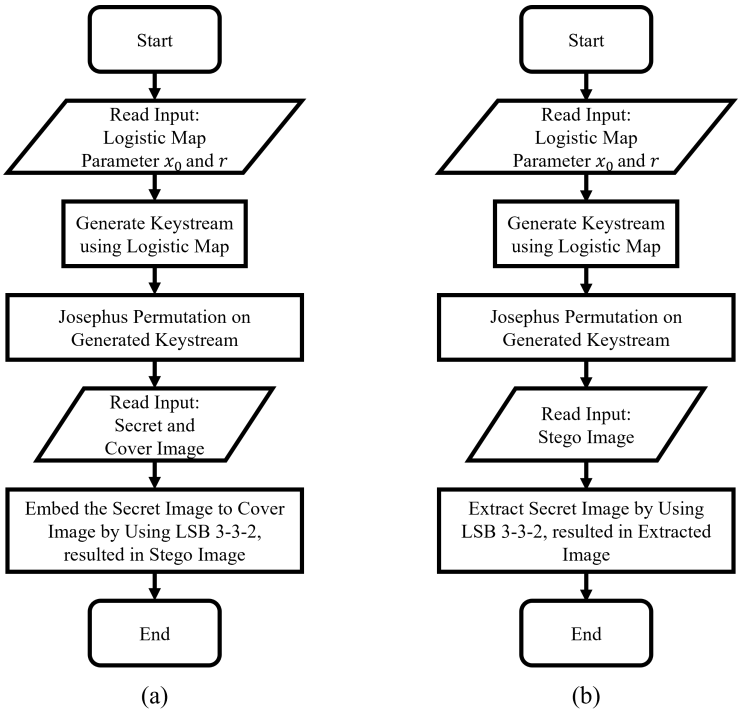


Figure 2. The embedding (**a**) and extraction algorithm (**b**).

3.1. Logistic Map

In this subsection, we demonstrate the use of the logistic map for generating the keystream in our steganographic algorithm. To ensure the chaotic behavior required for secure keystream generation, we first determine the appropriate range for the logistic map parameters using the Lyapunov exponent and bifurcation diagram. By carefully selecting these parameters, we can generate a highly unpredictable and secure keystream.

3.1.1. Lyapunov Exponent of the Logistic Map

Let us consider two iterations of the logistic map starting from two closely spaced values, namely, x_0 and $x_0 + \delta x_0$. These values will become x_1 and $x_1 + \delta x_1$, ..., $x_n + \delta x_n$. The expansion of $f(x)$ for x_n yields

$$\delta x_n = f'(x_{n-1})\delta x_{n-1} \tag{2}$$

assuming that δx_n is very small. Thus, the separation distance between the two values after n steps, δx_n, is related to their initial separation δx_0 as

$$\left|\frac{\delta x_n}{\delta x_0}\right| = \prod_{i=0}^{n-1}\left|f'(x_i)\right| \tag{3}$$

For large values of n, Equation (3) is expected to vary exponentially, like

$$\left|\frac{\delta x_n}{\delta x_0}\right| = e^{\lambda_L n}(\text{for large } n) \tag{4}$$

and therefore, the Lyapunov exponent λ_L is defined as

$$\lambda_L = \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}\ln\left|f'(x_i)\right| \tag{5}$$

If $\lambda_L > 0$, then the two values diverge from each other for large n, indicating chaos. However, if the two values converge to a fixed point or a limit cycle, they approach each other, corresponding to $\lambda_L < 0$.

Therefore, a system can be considered chaotic or not by examining the sign of the Lyapunov exponent. For $0 \leq r \leq 4$, the graph of the Lyapunov exponent for the logistic map is as follows.

As shown in Figure 3, the value of r indicating chaotic sensitive dependence is around $r = 3.5688$, as this is where the Lyapunov exponent becomes positive. After the value of r surpasses approximately 3.5688, the graph appears to be positive. Thus, values of r greater than that will be used as parameters for the logistic map.

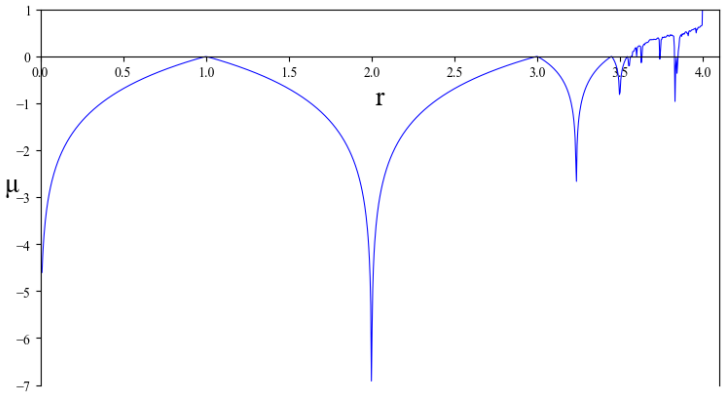


Figure 3. Lyapunov exponent graph of the logistic map growth rate for the range $0 \leq r \leq 4$.

3.1.2. Bifurcation Diagram of the Logistic Map

Along with sensitive dependence, a characteristic that must be present in a chaotic function is randomness or topological transitivity. This property can be observed using a bifurcation diagram.

Figure 4 shows the bifurcation diagram for the logistic map in the range $0 \leq r \leq 4$. For some small values of r, there is always a fixed point, namely, 0 for $r < 1$ and a non-zero value for $1 < r < 3$. This image only shows attractors, the set of x values where their iterations converge, for different values of r. There are also unstable fixed points and limit cycles that are not visible here. For example, the stable fixed point for $r < 1$ continues for $r > 3$ but becomes unstable, so its iteration does not persist and is not visible for $r > 3$. Instead, a stable attractor, with a cycle length of 2, emerges for r slightly greater than 3.

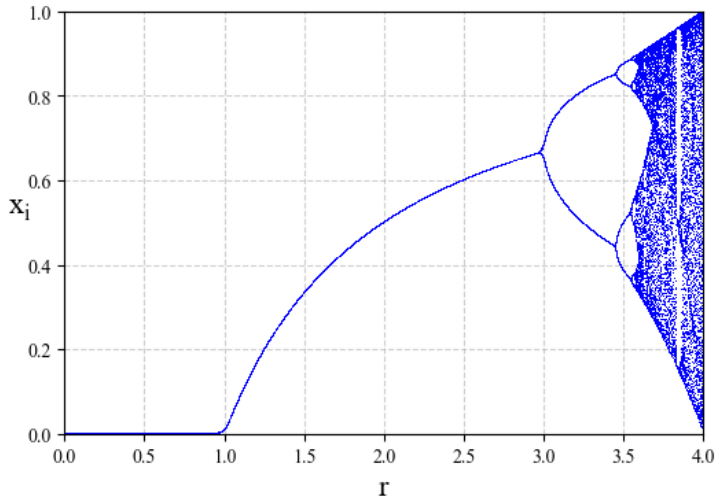


Figure 4. Bifurcation diagram of the logistic map growth rate for the range $0 \leq r \leq 4$.

In addition, the period-doubling cascade, a bifurcation event within bifurcation [23], can also be observed starting from $r = 3$ up to about $r = 3.5688$, where chaos occurs. Therefore, based on the analysis of the Lyapunov exponent and bifurcation diagram of the logistic map, the range of r values indicating chaotic behavior is estimated to be $3.5688 < r < 4$.

3.1.3. Keystream Generation with the Logistic Map

The keystream generation is crucial in determining the secrecy and security of this system; the calculation of the logistic map must be performed with high precision. This is to ensure that initial values with very small differences will yield significantly different extracted images, making them unrecognizable.

To control this precision level, the `decimal` library, which provides the `Decimal` data type, is used. This data type has fast and much more precise rounding than the `float` data type. The precision level of the calculations can be set using the command `getcontext().prec = n`, where `n` is the desired precision level. With this, the precision level can be adjusted, such as to the available memory, required security level, and so on.

The precision level is also directly related to the security level of the stego image. If the precision level used in the calculation is not high enough, then very small changes in the initial value will not reflect chaotic behavior. As a result, the system will be more vulnerable to brute force attacks. However, the high precision level must also not have significant drawbacks in terms of computation. This means that the precision level used should not excessively sacrifice computation time.

Based on Figure 5, the increase in calculation time with increasing precision is something that can be predicted. However, the range of the calculation time can be considered small, with the difference between the precision levels of 16 digits and 60 digits being about 0.10 s. Thus, this method can adjust to the needs of security levels without a drastic difference in time.

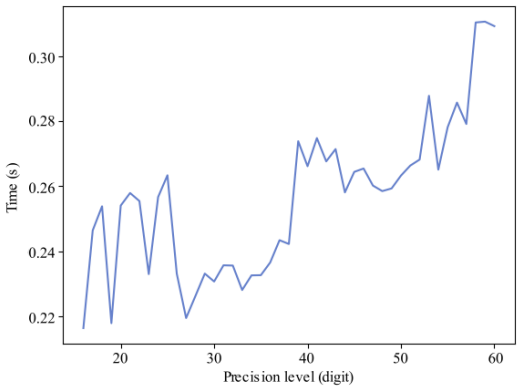


Figure 5. Time graph of keystream calculation with precision levels from 16 digits to 60 digits for a secret image of size 512 × 512 pixels.

To conclude, as shown in Figure 6, the keystream generation process begins with converting x_0 and r to `decimal` format for high-precision calculations. The initial value of x is set to x_0. The logistic map computation is then performed for a specified number of iterations, represented by `size`. If the final iteration is reached, the process proceeds to generate the keystream output. During each iteration, the new value of x is computed based on the logistic map equation. This value of x is then modified to represent a pixel location or a non-negative integer. Instead of storing each computed value in an array, which would increase the memory usage and execution time, a generator is used to temporarily collect the values of x into memory. Once all iterations are completed, the function outputs a generator containing the keystream.

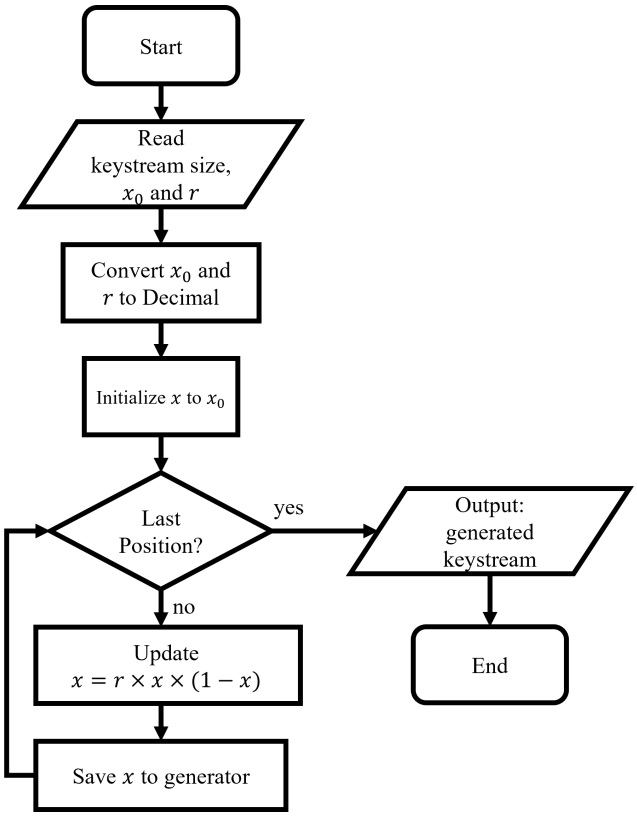


Figure 6. Keystream generation using logistic map.

3.2. Keystream Shuffling Using Josephus Permutation

In its implementation, the Josephus permutation is used to shuffle the keystream generated using the logistic map. This shuffling is performed for each element of the keystream, causing a change in the position of its indices. This shuffling will result in a new order of the keystream, which will later determine the positions for embedding pixels in the secret image into the cover image using the LSB 3-3-2 embedding technique. Algorithm 1 illustrates the basic logic of implementing the Josephus permutation on the keystream generated by the logistic map.

Algorithm 1 Josephus permutation.

Input: *secret_size*, *cover_size*
Output: *locations*
  locations ← []
  temp ← [0 . . . *cover_size*]
  chaos ← **get_chaos**(*secret_size* × 3)
  while len(*chaos*) > 0 **do**
    position ← *chaos*.***popleft***()
    position ← *position* % **len**(*temp*)
    locations.***append***(*temp*[*position*])
    delete *temp*[*position*]
  end while
  return *locations*

3.3. LSB 3-3-2 Embedding Technique

Assuming both the secret image and the cover image are in the RGB scheme, the LSB 3-3-2 embedding technique embeds the first 3 bits, the next 3 bits, and the last 2 bits of one color channel of the secret image into the last 3 bits of the red channel, the last 3 bits of the green channel, and the last 2 bits of the blue channel of the cover image, respectively. This is

where the LSB 3-3-2 embedding technique gets its name. However, this also means that the LSB 3-3-2 embedding technique must use an RGB scheme cover image rather than another scheme. Figure 7 illustrates the changes in a pixel in the cover image after embedding a red channel from a secret image. The changes are difficult to discern with the naked eye.

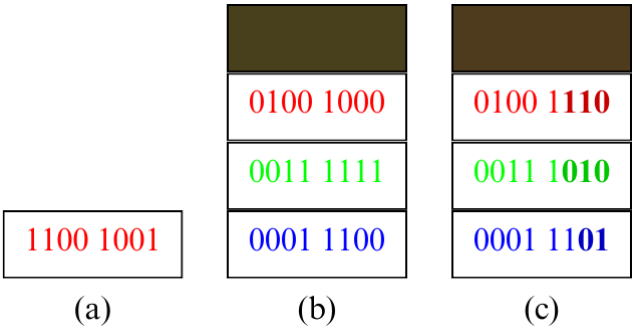


Figure 7. (**a**) Red channel of a pixel in the secret image, (**b**) composition of an RGB pixel in the cover image, (**c**) new RGB pixel resulting from embedding (**a**) into (**b**) using LSB 3-3-2.

The distribution of secret image bits into cover image bits is made this way because the human eye is most sensitive to changes in the blue channel [27]. Therefore, to minimize visual detection by the human eye, embedding in the blue channel is kept to a minimum while preserving the smallest possible quality changes in the red and green channels.

4. Evaluation and Discussion

To assess the effectiveness of the proposed RGB-to-RGB image steganography algorithm, we begin by detailing the materials and apparatus used for our experiments, including the specific images selected for both secret and cover images, which are listed in Tables 2 and 3. Following this, we conduct a comprehensive evaluation of the generated keystream, image quality, and processing time.

Table 2. Secret image data.

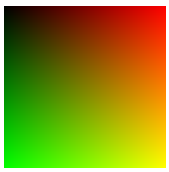
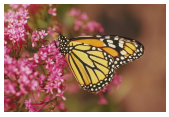
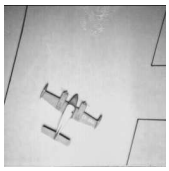
Name	Preview	Pixels	Size (KB)
rainbow		256 × 256	0.777
monarch		768 × 512	604
dipxe		320 × 138	5.43
plane		256 × 256	36

Table 3. Cover image data.

Name	Preview	Pixels	Size (KB)
lenna		512 × 512	462
malamute		1616 × 1080	2897
pepper		512 × 512	494
kid		487 × 703	595

4.1. Materials

The potential impact of texture, complexity, and color distribution on the quality and processing time of image steganography methods is significant. Images with complex textures can better mask embedded data, reducing noticeable artifacts, while simpler textures might reveal changes more easily. High-complexity images, with many variations and details, can hide data more effectively, whereas simpler images might make hidden data more detectable. Diverse color distributions can disperse hidden data more effectively, making detection harder, while uniform color images might show more noticeable changes. Processing time can also increase with more complex textures and high-complexity images due to the need to preserve intricate details.

As shown in Table 2, we have chosen a diverse set of secret images to ensure comprehensive evaluation. For example, Rainbow (256 × 256 pixels, 0.777 KB) has a wide range of colors and gradients, representing high color variability, and Monarch (768 × 512 pixels, 604 KB) has fine textures and intricate patterns, providing high complexity. Plane and Dipxe also feature varied complexities and color distributions, testing the embedding algorithm's robustness.

Similarly, our cover images shown in Table 3 exhibit a range of characteristics to test the robustness of our method. Lenna (512 × 512 pixels, 462 KB) has balanced color distribution and texture, while Malamute (1616 × 1080 pixels, 2897 KB) includes both smooth and textured areas. Pepper and Kid also provide varied textures and color distributions, enhancing the testing of our method's ability to handle of different cover image types.

4.2. Apparatus

The experiment and analysis were conducted using Python version 3.11 on a computer with the following specifications:

- Processor : 12th Gen Intel(R) Core(TM) i5-12400F
- Operating System : Windows 11 Home
- Memory : 16 GB RAM

4.3. Keystream Analysis

In this section, we examine the sensitivity of the keystream generated by the chaotic map. Next, we analyze the key space to demonstrate the algorithm's robustness against brute force attacks. Finally, we assess the randomness quality of the generated keystream.

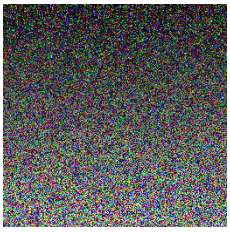
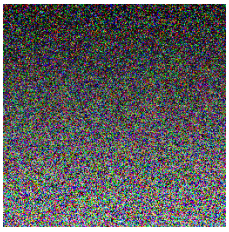
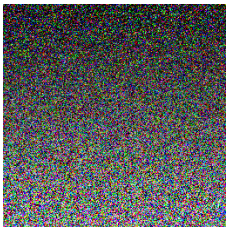
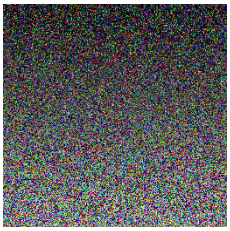
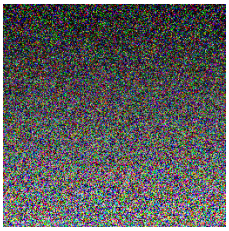
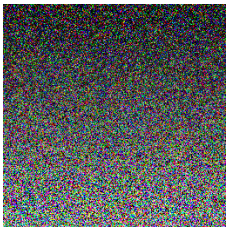
4.3.1. Key Sensitivity Analysis

The key, denoted as the initial value x_0 in the chaos calculation, plays a crucial role in this study. This experiment is designed to explore the impact of small differences between the key used for embedding and the one used for extraction on the similarity of the original secret and extracted images. A notable dissimilarity between the two images arising from a minor variance in the embedding and extraction keys signifies a more resilient system. Essentially, a random guess for the extraction key that is not identical to the one used for embedding, even if it is close, will yield a completely different extracted image compared to the original secret image.

The experiment commences by embedding the secret image into the cover image with the parameters $x_0 = 0.5$ and $r = 3.87$. Subsequently, the extraction process is repeated six times with different values of x_0 and a constant value of r. The secret and cover images employed for this analysis are named `rainbow` and `lenna`, respectively.

The six extraction results presented in Table 4 exhibit distinct images, even with variances at the precision level of 10^{-30}. Such pronounced differences at this `decimal` precision indicate the system's high sensitivity to the key. This sensitivity is further enhanced by the `decimal` library, which permits adjustments in the precision level used in calculations. This theoretically ensures the system's continual sensitivity to the key, provided that memory is consistently sufficient.

Table 4. Extraction results for six different x_0 values.

$x_0 = 0.5 + 10^{-15}$	$x_0 = 0.5 + 10^{-18}$	$x_0 = 0.5 + 10^{-21}$
		
$x_0 = 0.5 + 10^{-24}$	$x_0 = 0.5 + 10^{-27}$	$x_0 = 0.5 + 10^{-30}$
		

In addition to visual inspection, sensitivity can be quantified through Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR). To demonstrate this, we conduct a brute force experiment with a precision level of five digits, using the secret image `rainbow` and the cover image `lenna`. Embedding is performed with the values $x_0 = 0.5$ and $r = 3.7$, while hacking attempts cover the range $0.45 \leq x_0 \leq 0.56$ and $3.65 \leq r \leq 3.75$. The increment in values for x_0 and r for each brute force attempt is 0.01 or 10^{-2}, consistent with the precision level used. Verification of a successful hack involves comparing the PSNR value of the extracted secret image with the original; if the PSNR value is infinite

dB, then the extracted secret image from the brute force attempt is considered the target secret image.

Figure 8 illustrates the successful hacking of the image for the specific combination $x_0 = 0.5$ and $r = 3.7$, matching the configuration employed during the embedding process. For this particular combination, the MSE value is 0, and the PSNR value is infinite. In contrast, the MSE values for other combinations exceed 90, and the PSNR values for those combinations fall below 30 dB. It is evident that, except at the correct x_0 and r values, both MSE and PSNR values consistently provide similar error for other combinations. Consequently, this steganography method exhibits notable resistance to brute force attacks, demonstrating no indication that proximity in the x_0 and r values results in smaller MSE or higher PSNR.

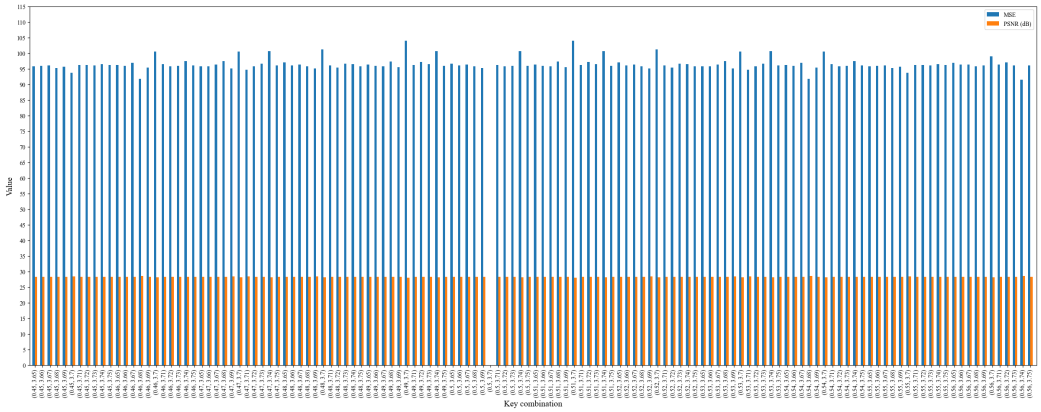


Figure 8. MSE and PSNR values for brute force experiment.

4.3.2. Key Space Analysis

The security level of this steganography method is directly contingent on the number of possible keys. A larger number of keys translates to more potential combinations, rendering it challenging for an attacker to exhaustively test all key combinations through brute force or similar methods. Key space analysis involves evaluating the overall number of potential keys and the difficulty associated with testing each key.

In this study, the keys are denoted as x_0 and r, both serving as inputs to the chaotic logistic map function. The domain for each key is defined as $x_0 \in [0, 1]$ and $r \in (3.6, 4.0)$. Unlike the `float` data type, which can be easily determined using the `sys.float_info` command in Python, x_0 and r are specialized data types defined in the `decimal` library. This `decimal` data type permits adjustable precision, with the largest precision being 4.25×10^8 for a 32-bit machine and 9.99×10^{17} for a 64-bit machine. However, the feasible precision levels are contingent on the available memory on the machine. For the purposes of analysis, the precision level employed in the key space calculation aligns with the precision used in the embedding and extraction processes, which is 20 digits.

In the calculation, it is necessary to define a granularity or step for each element in x_0 and r. Similar to the key sensitivity analysis, x_0 exhibits a significant impact even at a precision level as small as 10^{-30}. Consequently, it can be assumed that r shares the same sensitivity, given that it is of the same data type. Hence, the granularities Δx and Δr, for this example, are set at 10^{-30}. The total key space is determined by the number of possible values for each parameter, formulated as follows:

$$\begin{aligned} \text{Key Space} &= (x_0 \text{ count} \times r \text{ count})^{\text{Precision}} \\ &= \left(\frac{(1-0)}{10^{-30}} \times \frac{(4-3.6)}{10^{-30}} \right)^{20} \\ &= 4 \times 10^{1080} \end{aligned} \tag{6}$$

If an attacker has a machine capable of testing one billion (10^9) combinations per second, the estimated time required to successfully find the key is about 4×10^{1071} s, which is more than 10^{1062} centuries. However, this is a theoretical calculation and depends on many factors such as algorithm efficiency, the device used, and the security system of the system. Nevertheless, given the assumptions, this calculation provides a rough estimate of the scale involved.

4.3.3. Key Randomness Analysis

The keystream generated using the chaotic logistic map function is then subjected to randomness analysis to verify its randomness after being subjected to the Josephus permutation. This testing is performed using the NIST (National Institute of Standards and Technologies) SP 800-22 Test Suite. Research conducted by Bassham et al. [28] lists fifteen types of statistical tests, namely, the following:

1. Frequency;
2. Block Frequency;
3. Cumulative Sums;
4. Runs;
5. Longest Run;
6. Rank;
7. FFT;
8. Non-Overlapping Template;
9. Overlapping Template;
10. Universal;
11. Approximate Entropy;
12. Random Excursions;
13. Random Excursions Variant;
14. Serial;
15. Linear Complexity;

Randomness analysis of the keystream will be conducted with the values $x_0 = 0.5$, $r = 3.87$, and a precision level of five digits. Table 5 shows the results of randomness testing of the shuffled keystream with a confidence level of $\alpha = 0.1$. The fifteen types of tests indicate that the keystream from the logistic map that is shuffled by the Josephus permutation is still random.

Table 5. Results of randomness test of keystream.

No.	Test Type	P_{value}	Proportion	Result
1.	Frequency	0.534146	1	Pass
2.	Block Frequency	0.350485	0.9	Pass
3.	Cumulative Sums	0.534146	1	Pass
4.	Runs	0.739918	1	Pass
5.	Longest Run	0.911413	1	Pass
6.	Rank	0.534146	1	Pass
7.	FFT	0.213309	1	Pass
8.	Non-Overlapping Template	0.739918	1	Pass
9.	Overlapping Template	0.911413	1	Pass
10.	Universal	0.000000	1	Pass

Table 5. *Cont.*

No.	Test Type	P_{value}	Proportion	Result
11.	Approximate Entropy	0.066882	1	Pass
12.	Random Excursions	-	1	Pass
13.	Random Excursions Variant	-	1	Pass
14.	Serial	0.534146	0.9	Pass
15.	Linear Complexity	0.350485	1	Pass

It can also be concluded from Table 5 that using the Josephus permutation to shuffle a random keystream does not induce a recognizable pattern in it; hence, the randomness stays present. This acts as an extra layer of security that attackers have to go through before obtaining our secret image.

4.4. Image Quality Analysis

In this section, we examine the payload that can be accommodated by the algorithm. Further, we also measure the quality of the stego and the extracted image by comparing them with cover and secret image, respectively.

4.4.1. Payload Analysis

Payload refers to the amount of secret information that can be embedded into a cover image [29]. This quantity largely depends on the embedding technique and the structure of the cover image. For instance, with the LSB technique, the payload can be calculated mathematically. The fewer cover image pixels required to embed one secret image pixel, the larger the maximum payload of the LSB embedding.

In LSB-1, $24/1 = 24$ pixel channels are required to accommodate every $8 \times 3 = 24$ bits of the secret image. Thus, it requires $24/3 = 8$ RGB pixels of the cover image for each secret image RGB pixel. In LSB-2, $24/2 = 12$ pixel channels are needed to accommodate every 24 bits of the secret image, requiring $12/3 = 4$ RGB pixels of the cover image for each secret image RGB pixel. In LSB-4, $24/4 = 6$ pixel channels are necessary to accommodate every 24 bits of the secret image, requiring $6/3 = 2$ RGB pixels of the cover image for each secret image RGB pixel. Finally, for LSB 3-3-2, it requires one pixel to accommodate one pixel channel (leaving one channel unused for the third pixel) of the secret image, thus requiring three pixels of the cover image for each pixel for each secret image RGB pixel.

Therefore, for an RGB secret image scheme, the LSB 3-3-2 embedding technique has a payload approximately 2.67 times larger than LSB-1 and approximately 1.33 times larger than LSB-2. While LSB 3-3-2 has a payload 1.5 times smaller than LSB-4, this smaller payload is compensated for by the higher quality of the stego image.

4.4.2. Cover vs. Stego Image Comparison

An effective image steganography system should generate stego images that appear visually indistinguishable from their cover images, rendering the differences imperceptible to the human eye as illustrated in Table 6. To discern changes at the pixel level, the images in Table 6 are magnified and presented in Table 7. Upon magnification, differences become apparent, with observed variations in pixel values between the stego and cover images. Nevertheless, these subtle alterations are unlikely to raise suspicion for a human observer. This aligns with the MSE and PSNR results detailed in Tables 8 and 9, respectively.

Table 6. Cover image quality vs. stego image quality.

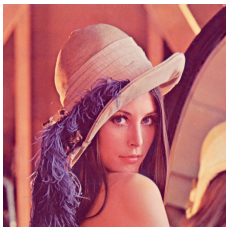
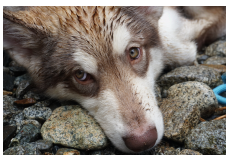
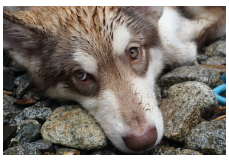
Scenario	Cover Image	Stego Image
rainbow to lenna		
rainbow to malamute		
monarch to malamute		

Table 7. Cover image quality vs. stego image quality (zoomed in).

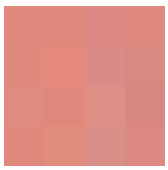
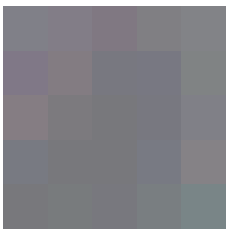
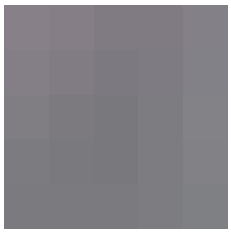
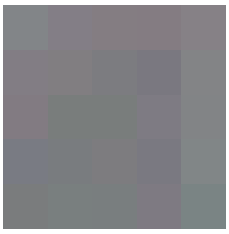
Scenario	Cover Image	Stego Image
rainbow to lenna		
rainbow to malamute		
monarch to malamute		

Table 8. MSE cover image vs. stego image.

Scenario	MSE			
	R	G	B	Aggr
rainbow to lenna	9.6972	9.6139	2.1368	7.1493
rainbow to malamute	1.4398	1.4464	0.3195	1.0686
monarch to malamute	5.8066	7.0549	1.6699	4.8438

Table 8 shows the MSE between the cover image (left side of the arrow) and the stego image (entire scenario) for each color channel of the pixels and the aggregate. In each scenario, the MSE for each channel is small, i.e., below 1.0. Similarly, the aggregate MSE is below 10.0. This means that pixel values in the stego image have very small differences from the pixel values in the cover image.

Table 9. PSNR cover image vs. stego image.

Scenario	PSNR			
	R	G	B	Aggr
rainbow to lenna	49.2779	49.4613	56.4632	39.5926
rainbow to malamute	65.8861	65.8258	73.0960	47.8428
monarch to malamute	54.3625	71.5566	82.8429	41.2790

Table 9 shows the PSNR between the cover image and the stego image for each color channel of the pixels and the aggregate. In each scenario, the PSNR for each channel is very good, i.e., above 40 dB. Similarly, the aggregate PSNR is above 40 dB for all but one scenario. This means that, similar to the MSE results, pixel values in the stego image have very small differences from the pixel values in the cover image.

Table 10 shows a more thorough MSE and PSNR analyses between for several other images and with different embedding methods. For this, a comparison with the LSB-1, LSB-2, LSB-4 [11], and LSB 3-3-2 is conducted. We can see that similar results are obtained as shown in Table 9.

We can also see that the LSB 3-3-2 embedding technique has worse MSE and PSNR values compared to LSB-1 and LSB-2 but is better than LSB-4. This is because LSB-1 and LSB-2 only embed the bits of the secret image into the last one and two bits of the cover image pixels, respectively, which can result in stego images with less noticeable visual differences.

Table 10. Comparison of MSE and PSNR against other embedding techniques.

Cover Image	Secret Image	LSB Pattern	MSE	PSNR
pepper	plane	LSB-1 [11]	0.3329	52.9064
		LSB-2 [11]	0.8430	48.8721
		LSB-4 [11]	7.1273	39.6015
		LSB 3-3-2	2.1217	44.8639
	dipxe	LSB-1 [11]	0.2243	54.6222
		LSB-2 [11]	0.5713	50.5615
		LSB-4 [11]	4.7631	41.3518
		LSB 3-3-2	1.2461	47.1751
kid	plane	LSB-1 [11]	0.2553	54.0600
		LSB-2 [11]	0.6351	50.1017
		LSB-4 [11]	5.3143	40.8763
		LSB 3-3-2	1.5481	46.2327
	dipxe	LSB-1 [11]	0.1724	55.7646
		LSB-2 [11]	0.4260	51.8364
		LSB-4 [11]	3.5561	42.6210
		LSB 3-3-2	0.9273	48.4584

4.4.3. Secret vs. Extracted Image

In addition to generating a stego image that closely resembles the cover image, a proficient image steganography algorithm must ensure the extraction process yields an image similar or identical to the original secret image. Table 11 presents the Mean Squared Error (MSE) between the original secret image and the extracted secret image for each color channel of the pixels as well as the aggregate. In each case, the MSE for every channel and the aggregate is 0.0, signifying that the extracted secret image remains entirely undistorted.

Table 11. MSE original secret image vs. extracted secret image.

Scenario	MSE			
	R	**G**	**B**	**Aggr**
rainbow from lenna	0.0	0.0	0.0	0.0
rainbow from malamute	0.0	0.0	0.0	0.0
monarch from malamute	0.0	0.0	0.0	0.0

Table 12 shows the PSNR between the original secret image and the extracted secret image for each color channel of the pixels and the aggregate. In each scenario, the PSNR for each channel and the aggregate is infinite. This means that, similar to the MSE results, the extracted secret image is not distorted at all. Therefore, we consider the visual inspection of the quality of the secret image and the extracted secret image to not be necessary.

Table 12. PSNR original secret image vs. extracted secret image.

Scenario	PSNR			
	R	**G**	**B**	**Aggr**
rainbow from lenna	∞	∞	∞	∞
rainbow from malamute	∞	∞	∞	∞
monarch from malamute	∞	∞	∞	∞

4.5. Time Analysis

4.5.1. Keystream Generation Time Analysis

This analysis only assesses the time taken to generate a keystream for each secret image and cover image pair. Seven iterations of this analysis are performed, and the average time of the seven yields aggregate results. The parameters for the chaotic function which is used to generate the keystream are $x_0 = 0.5$ and $r = 3.87$, and will remain constant throughout all iterations. The precision level for generating the keystream is up to 20 decimal places.

Table 13 shows an increase in the keystream generation time when going from embedding `rainbow` to `monarch`. This is consistent with the nature of the keystream generation algorithm, as it only takes into account the size of the secret image. The increase is also linear $O(n)$, as the size of `monarch` is six times that of `rainbow`, which can be seen in Table 2.

Table 13. Average keystream generation time.

Cover Image	Secret Image	Keystream Generation Time (s)
lenna	rainbow	0.0606
malamute	rainbow	0.0612
malamute	monarch	0.3643

4.5.2. Keystream Shuffling Time Analysis

This analysis only assesses the time taken to shuffle a keystream for each secret image and cover image pair. As with the keystream generation time analysis, this keystream shuffling time analysis has seven iterations, and the average time of the seven yields aggregate results. The parameters for the chaotic function and precision level are not changed and remain constant throughout all iterations.

Similar to Table 13, Table 14 also shows an increase in the keystream generation time when going from embedding `rainbow` to `monarch`. While the reason for this is the same as the previous analysis, one might question why the time to shuffle the keystream takes longer than to generate it. One of the reasons why this process takes a long time is because of the deletion operation on a `dequeue` data structure. This can be seen in Algorithm 1, where the said use of `deque` is performed.

Table 14. Average keystream shuffling time.

Cover Image	Secret Image	Keystream Shuffling Time (s)
lenna	rainbow	0.8836
malamute	rainbow	0.8988
malamute	monarch	5.2078

4.5.3. Embedding and Extraction Time Analysis

The embedding process in this experiment is limited to the insertion of a secret image and excludes the generation of pixel distribution locations via the chaotic logistic map function and keystream shuffling through the Josephus permutation, as they have previously been performed separately. Similarly, the extraction process involves retrieving a secret image from a cover image using the LSB 3-3-2 embedding technique, with the pixel distribution locations retained without recalculation.

In the same way as the previous time analyses, this analysis also comprises seven iterations, each involving the insertion and extraction of a secret image into and from a cover image. The average time taken across these seven iterations will yield aggregate results. The parameters and precision will also stay the same as those of the previous analysis.

Table 15 demonstrates that the insertion and extraction times are solely contingent on the size of the secret image, remaining unaffected by the size of the cover image. Notably, there is a noteworthy escalation in the insertion and extraction times for varying secret image sizes.

Table 15. Average insertion and extraction times.

Cover Image	Secret Image	Embed Time (s)	Extract Time (s)
lenna	rainbow	0.8172	0.7864
malamute	rainbow	0.7946	0.7634
malamute	monarch	22.8363	22.7879

5. Conclusions

In this research, we introduce a novel RGB-to-RGB image steganography algorithm that incorporates the Josephus permutation into the LSB 3-3-2 technique. While the Josephus permutation has been extensively utilized in image cryptography, its application in image steganography is novel. Our study differentiates itself by leveraging the Josephus permutation to enhance the randomness and security of the embedding process. Previous studies have primarily focused on embedding text or grayscale images using various LSB schemes. However, the use of basic logistic maps in LSB techniques has shown vulnerabilities to steganalysis due to their predictable patterns, highlighting the need for increased randomness to ensure better security.

Our implementation initially employed the chaotic logistic map function within the range $3.5688 < r < 4$ to generate the keystream. The effectiveness of this keystream generation was assessed using the Lyapunov exponent and bifurcation diagram, ensuring the parameters induced chaotic behavior. The generated keystream was then scrambled using the Josephus permutation, providing enhanced randomness. This step significantly mitigated the risk of detection by steganalysis, as the Josephus permutation introduced non-repetitive sequences, improving the scrambling effects and permutation efficiency.

Our results demonstrate that even a slight difference between the key used for embedding and the one used for extraction results in a completely different extracted image compared to the original secret image. This property enhances the security of the technique against brute force attacks. The randomly shuffled key sequence from the Josephus permutation was employed to determine pixel distribution locations in both the embedding and extraction processes using the LSB 3-3-2 embedding technique.

For RGB secret image schemes, the LSB 3-3-2 technique yields a payload approximately 2.67 times larger than LSB-1 and around 1.33 times larger than LSB-2. Although LSB 3-3-2 has a payload size 1.5 times smaller than that of LSB-4, this reduced payload is compensated for by the higher quality of the stego image. Our analysis of the Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) between the stego and cover images reveals satisfactory results (MSE below 10.0 and PSNR above 40 dB). For the secret versus extracted images, the MSE is 0.0 and the PSNR is infinite, indicating successful extraction with high fidelity.

By addressing these gaps, our proposed method advances the field of image steganography by providing a more secure and efficient technique for hiding color images. This research offers practical implications for its application in secure communications, ensuring that the embedded data remain well protected against various types of attacks.

6. Future Works

While our RGB-to-RGB image steganography algorithm exhibits security and ease of implementation, we acknowledge the limitations in the variety of image samples used in our current experiments. To address this, we plan to conduct further testing with a broader range of images, including those with varying levels of texture, complexity, and color distributions. This will allow us to evaluate the impact on image quality using metrics such as MSE, PSNR, and processing time, thereby providing a more comprehensive assessment of the proposed steganographic method. Additionally, we recognize the lack of performance comparisons with other methods under similar payload conditions and plan to include such comparisons in our future work to better assess the advantages of our method.

In addition to these improvements, we aim to develop a more intricate yet efficient Josephus permutation to generate key sequences that are even more random within a shorter timeframe. We will explore alternative chaos functions to determine the optimal performer when its key sequence undergoes shuffling through the Josephus permutation. Investigating embedding techniques beyond LSB that can yield stego images with larger and higher-quality payloads is also a priority. Furthermore, optimizing embedding and extraction times for secret images of different sizes by leveraging faster programming languages such as C++ and employing parallelization techniques will be explored. Finally, to address issues related to contemporary information exchange, which often relies on compression techniques, we aim to incorporate countermeasures to ensure the integrity of extracted images.

Author Contributions: Conceptualization, M.R.Y. and M.F.S.; Formal analysis, M.R.Y., S.M. and M.F.S.; Funding acquisition, C.A.; Investigation, M.R.Y.; Methodology, M.R.Y. and M.F.S.; Software, M.R.Y.; Supervision, M.F.S.; Writing—original draft, M.R.Y.; Writing—review and editing, M.R.Y., M.F.S. and C.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by University of Indonesia, grant number NKB-804/UN2.RST/HKP.05.00/2023.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable.

Acknowledgments: Universitas Indonesia supports this research through the International Indexed Publication (PUTI) Q2 Grant, 2023.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Artz, D. Digital steganography: Hiding data within data. *IEEE Internet Comput.* **2001**, *5*, 75–80. [CrossRef]
2. Cyber, B.G. Steganography: Hidden in Plain Sight, 2023.
3. Yang, C.H.; Weng, C.Y. A novel steganographic method based on JPEG and quantization table modification. *Signal Process.* **2008**, *88*, 2134–2140.
4. Xu, G.; Ping, X.; Su, Y. Secure and robust image steganography based on logistic chaotic map and DNA sequence operations. *Multimed. Tools Appl.* **2019**, *78*, 13203–13225.
5. El-Sayed, A.M. The Josephus permutation applied to cryptographic algorithms. *J. Inf. Secur. Appl.* **2015**, *22*, 30–39.
6. Meng, Q.; Yang, C. Image encryption algorithm based on the Josephus problem and self-adaptive encryption. *Opt. Lasers Eng.* **2017**, *96*, 43–50.
7. Subramanian, A.M.S.B.A.; Elharrouss, N. Image steganography: A review of the recent advances. *IEEE Access* **2021**, *9*, 23409–23423. [CrossRef]
8. Bhuiyan, T.; Sarower, A.H.; Karim, R.; Hassan, M. An image steganography algorithm using LSB replacement through XOR substitution. In Proceedings of the IEEE International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 24–25 July 2019; pp. 44–49.
9. Patra, A.S.; De, D. A new high-capacity LSB image steganography method combined with DES encryption and logistic map. *J. King Saud Univ. Comput. Inf. Sci.* **2023.** [CrossRef]
10. Elharrouss, A.; Al-Maadeed, N.; Al-Maadeed, S. An image steganography approach based on k-least significant bits (k-lsb). In Proceedings of the 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2–5 February 2020.
11. Astuti, N.D. Implementasi Kriptografi Berbasis Chaos dan Steganografi Menggunakan Teknik Penyisipan LSB pada Citra Digital. Undergraduate Thesis, Universitas Indonesia, Kota Depok, Indonesia, 2019.
12. Fridrich, J.; Goljan, M.; Du, R. Reliable Detection of LSB Steganography in Color and Grayscale Images. In Proceedings of the ACM Workshop on Multimedia and Security, Ottawa, ON, Canada, 5 October 2001; pp. 27–30.
13. Wang, X.; Liu, L. Application of chaotic Josephus scrambling and RNA computing in image encryption. *Multimed. Tools Appl.* **2021**, *80*, 23337–23358. [CrossRef]
14. Wolfram MathWorld. Josephus Problem, 2023. Available online: https://mathworld.wolfram.com/JosephusProblem.html (accessed on 11 August 2024).
15. Wang, Z.Y.; Zhu, X. An image encryption algorithm based on josephus traversing and mixed chaotic map. *IEEE Access* **2018**, *6*, 23733–23746. [CrossRef]
16. Hua, Z.; Xu, B.; Jin, F.; Huang, H. Image encryption using josephus problem and filtering diffusion. *IEEE Access* **2019**, *7*, 8660–8674. [CrossRef]
17. Liang, Z.C.; Qin, Z.; Qin, Q. An image encryption algorithm based on fibonacci q-matrix and genetic algorithm. *Neural Comput. Appl.* **2022**, *34*, 19313–19341. [CrossRef]
18. Rachmawanto, E.H.; Zulfiningrum, R. Medical image cryptosystem using dynamic josephus sequence and chaotic-hash scrambling. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 6818–6828.
19. Wu, X.; Zhang, X.; Liu, Z. Image steganography using logistic map and bit-plane complexity segmentation. *J. Real-Time Image Process.* **2022**. [CrossRef]
20. Li, T.Y.; Yorke, J.A. Period three implies chaos. *Am. Math. Mon.* **1975**, *82*, 985–992. [CrossRef]
21. Glass, L.; Mackey, M.C. *From Clocks to Chaos: The Rhythms of Life*; Princeton University Press: Princeton, NJ, USA, 1988.
22. Gleick, J. *Chaos: Making a New Science*; Viking: New York, NY, USA, 1987.
23. Peter. The Logistic Map. Available online: https://en.wikipedia.org/wiki/Logistic_map (accessed on 7 August 2024).
24. Kocarev, S.L.; Ljupco, S. *Chaos-Based Cryptography*; Springer: Berlin/Heidelberg, Germany, 2011.
25. Ruelle, D. *Chaotic Evolution and Strange Attractors*; Cambridge University Press: New York, NY, USA, 1989.
26. Hollar, D.W. Nonlinear maps and chaos. In *Magillâs Survey of Science: Physical Science*; Magill, F.N., Tombrello, T.A., Eds.; Salem Press: Pasadena, CA, USA, 1992; pp. 1556–1563.
27. Abdullah, A.H.; Hikmat, A. New approaches to encrypt and decrypt data in image using cryptography and steganography algorithm. *Int. J. Comput. Appl.* **2016**, *143*, 11–17.

28. Bassham, L.; Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Leigh, S.; Levenson, M.; Vangel, M.; Heckert, N.; Banks, D. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2010.
29. Setiadi, D.R.I.M.; Rustad, S. Digital image steganography survey and investigation (goal, assessment, method, development, and dataset). *Signal Process.* **2023**, *206*, 108908. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.