

Article

Fast and Lightweight Hybrid Image Encryption and Steganography Leveraging an SPN, Chaotic Maps, and LSB Substitution

Abdullah Alaklabi ^{1,2} , Muhammad Asfand Hafeez ³  and Arslan Munir ^{3,*} ¹ Department of Computer Science, Kansas State University, Manhattan, KS 66506, USA; alaklabi1@ksu.edu² Department of Computer Science, Shaqra University, Shaqra 11961, Saudi Arabia³ Department of Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA; mhafeez2024@fau.edu

* Correspondence: arslanm@fau.edu

Abstract

The rapid growth of digital communication has heightened the need for the secure transfer of sensitive image data. This is due to the increasing threats posed by cyberattacks and unauthorized access. Traditional encryption methods, while effective for text and binary data, often face significant challenges when applied to images, due to their larger size and complex structure. These characteristics make it difficult to provide a robust security solution. In this paper, we present a fast and efficient hybrid image encryption and steganography algorithm that leverages a substitution–permutation network (SPN), a chaotic logistic map (CLM), and least-significant-bit (LSB) substitution. This approach aims to improve data security and confidentiality while maintaining low computational complexity. The chaotic map generates random sequences for substitution and permutation, ensuring high unpredictability. The SPN framework improves the confusion and diffusion properties of the encryption process. The LSB substitution method hides the encrypted data values within the pixels of the cover image. We evaluate the security and efficiency of the proposed algorithm using various statistical tests, including measurement of the mean square error (MSE) and peak signal-to-noise ratio (PSNR) and pixel difference histogram (PDH) analysis. The results indicate that our algorithm outperforms many existing methods in terms of speed and efficiency, making it suitable for real-time hybrid encryption and steganography applications.

Keywords: chaotic map; image encryption; image steganography; least-significant-bit substitution; security; substitution–permutation network

1. Introduction

The secure transmission of images across public networks is becoming increasingly critical due to the rise of cyberattacks and unauthorized access to visual information. With the rapid expansion of image exchange across personal, corporate, and governmental settings, the risks of interception, manipulation, and exposure have become increasingly significant. These threats motivate the development of protection strategies that maintain confidentiality, integrity, and authenticity, especially for image data. Unlike text, images have a large size and exhibit redundancy and strong interpixel correlations; therefore, a direct transfer of techniques designed for text encryption is often inefficient.

DES and AES are examples of classical block ciphers that are essential for secure communication. However, when they are used directly to encrypt images, they typically fail



Received: 11 December 2025

Revised: 22 January 2026

Accepted: 30 January 2026

Published: 9 February 2026

Copyright: © 2026 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 \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

to exploit intrinsic image statistics and the parallel structure of pixel arrays [1–3]. To address these limitations, image focused cryptosystems have emerged around two interrelated approaches. The first approach is based on chaotic dynamics. Their sensitivity to initial conditions and ergodic behavior yield unpredictable, key dependent sequences suitable for diffusion and keystream generation [4,5]. The second approach is the substitution and permutation network, which realizes Shannon’s principles of confusion and diffusion. This is achieved through iterated S-box and P-box operations [6–9].

These approaches are often integrated by using chaotic maps such as a chaotic logistic map (CLM) to parameterize or key the components of the substitution and permutation network. This integration produces lightweight schemes with strong resistance to cryptanalysis [7,10]. To achieve higher security levels, researchers have explored modified logistic maps and higher dimensional chaotic systems. These enhancements increase the key sensitivity and further strengthen the system’s unpredictability. At the same time, surveys continue to organize and synthesize this approach [3].

There is a large amount of data being generated and transmitted due to the current advancements in data digitization. This makes the storage and transmission of sensitive data over open and unsecure communication channels a complex challenge [11]. Steganography, the process of hiding data within cover media such as images, audio, and video files, offers an effective solution to this challenge. It conceals secret information, rendering it undetectable to unauthorized individuals or attackers [12].

The different steganography schemes can be mainly divided into three main domains. These are the spatial, transform, and adaptive domains. The spatial domain, being the simplest, involves the modification of the pixel values of the cover image without considering the features and textures of the images. This results in a uniform capacity rate of data hiding across the whole cover image regardless of how smooth or complex an area of the cover image is [13]. The least significant bit (LSB) method is a spatial domain method that modifies the least significant bits of pixel values, making it a lightweight and efficient method for embedding encryption keys or other data within an image without significantly altering its appearance [14].

Hiding information within the least significant bit (LSB) of the steganography image is a particularly easy and common practice in image steganography [15,16]. This method changes the least significant bit planes of the image pixels so that no obvious difference is encountered, except for a tonal variation which is not discernable to the human eye. Usually, the process requires the R, G, and B channels of the images, with an interval range of 0–255 and a depth of 8×3 bits per pixel, offering more space for data to be hidden. After the RGB channels from the cover image have been separated and the secret data bits have been embedded into the LSB of any of the channels, the channels can be merged together to form the final steganography image, effectively hiding the data.

On the other hand, the Pixel Value Differencing (PVD) method exploits the advantages of the human visual system (HVS) by embedding information using the difference between the consecutive pixels of the cover image. This makes it possible for the PVD method to be capable of embedding data on the smooth and edged regions of the image with low and high embedding capacities, respectively. It is also able to keep the visual content intact, thereby improving secrecy.

Such spatial domain techniques, for example, image LSB substitution and PVD techniques, are highly sought after because of the ease they provide when embedding a lot of information within a cover image that does not change much visually. More specifically, PVD methods work better because of the nature of their approach, which is to embed information within the cover image areas of less detailed pixel intensity variations. Throughout the history of the development of steganography methods, both LSB substitution and PVD

have retained their relevance and are still very much in demand thanks to their effectiveness, speed, and low influence on the perceptibility of images, which means safety for the pertinent information within.

The transform domain of steganography involves the transformation of the cover image into the frequency domain using mathematical functions such as the Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT), or Discrete Fourier Transform (DFT). The embedding process involves the manipulation of the frequency components of the cover image instead of the image itself. The secret data is embedded into specific frequency coefficients, which are usually in the middle or high frequency ranges [17]. The transform domain usually achieves a high level of imperceptibility because modifications in this domain are not very noticeable to the human eye. The transform domain techniques are also highly resistant to steganalysis attacks as they are robust against many image processing operations, including scaling, compression, and noise addition [18].

DCT-based steganography is one of the more common transform domain techniques and is frequently applied to images. It involves the embedding of data into quantized DCT coefficients, focusing on the frequency components that have much less impact on the visual quality of the image. The DWT is another transform based technique where an image is decomposed into multiple frequency sub-bands (low and high frequencies). The data embedding is performed in the high frequency sub-bands which are less susceptible to detection through the human visual system [19,20].

The field of image encryption has undergone significant evolution over the years, with numerous algorithms being developed to enhance both security and efficiency. This literature review examines various encryption techniques, with a focus on those that incorporate chaotic maps, substitution–permutation networks (SPNs), and least-significant-bit (LSB) substitution. It also illustrates the evolution of image encryption and steganography techniques from traditional methods to advanced chaotic and combined approaches. These advancements not only enhance data security against a wide range of attacks but also address the need for maintaining visual integrity in encrypted images.

This paper presents an innovative hybrid approach that merges encryption and steganography through the implementation of an SPN, chaotic maps, and LSB substitution. We develop a fast, efficient, and secure steganographic method that first encrypts the data and then embeds the encrypted data within an image while maintaining the image's imperceptibility and integrity. We evaluate the effectiveness of the proposed methodology by conducting comprehensive experiments utilizing a range of performance metrics, including the mean square error (MSE) and peak signal-to-noise ratio (PSNR), and pixel difference histogram (PDH) analysis to quantify distortion, perceptual quality, and statistical regularity, respectively. The key contributions are as follows:

- We develop a hybrid approach that integrates encryption and steganography by utilizing a combination of SPN, CLM, and LSB substitution techniques. The integration of these methods results in a solution characterized by enhanced efficiency, robust security, and significant resilience against statistical attacks.
- In the proposed method, we introduce an SHA2-256 hash to ensure that the encryption algorithm responds to even minor changes in the plaintext. A small modification in plaintext results in a different hash value, which affects all encryption and steganography processes.
- The proposed algorithm allows the embedding of various data types, including text, images, and audio, by reshaping them into 2D matrices. This transformation is essential for leveraging a lightweight and efficient encryption algorithm through optimized matrix operations, thereby enhancing performance and scalability.

- We also perform a detailed evaluation of the proposed hybrid data encryption and image steganography algorithm, comparing it with other methods in the literature. The results show significant improvements in execution time, MSE, and PSNR, highlighting the method's superior performance and effectiveness.

The remainder of the paper is organized as follows: Section 2 presents the proposed hybrid encryption and steganography method. Section 3 reports the experimental results. Section 4 concludes the paper.

2. Methodology

The proposed hybrid image encryption and steganography scheme, illustrated in Figure 1, comprises various steps that collectively ensure seamless and secure communication. First, the plaintext message (image in Figure 1) is encrypted using a CLM and SPN to protect its confidentiality. The encrypted data is concatenated with the plaintext data header, which contains the data name, data size, and any additional metadata. The cover image, which embeds the encrypted image in our approach, is first pixel shuffled. Then, concatenated data (encrypted image and header) is embedded into the cover image via LSB substitution. Next, an inverse pixel shuffle is performed, creating the steganographic image. On the receiving side, these steps are reversed to extract the original plaintext image from the steganographic image. First, the steganographic image is pixel shuffled, and the plaintext header and encrypted plaintext data are extracted. Afterwards, the extracted encrypted data is decrypted using an SPN and CLM to yield the plaintext image. The following subsections provide details of each step of our proposed approach.

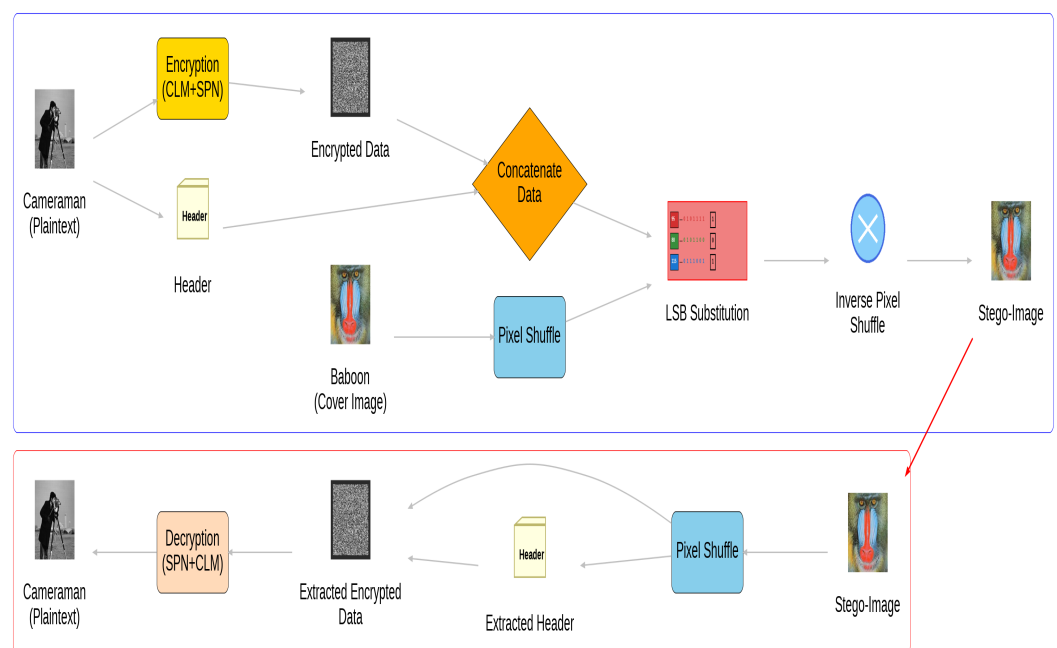


Figure 1. Overview of the proposed hybrid encryption and steganography scheme.

2.1. Encryption and Decryption Process

The encryption process employs a computationally efficient algorithm that utilizes a CLM to generate pseudorandom values, facilitating permutations of rows and columns. The SPN improves security through transformation operations that ensure confusion and diffusion, strengthening resistance against cryptanalysis. The final ciphertext is embedded into a cover image using the steganographic technique detailed in the following section.

2.1.1. Encryption with a Chaotic Logistic Map

The chaotic logistic map (CLM), known for its sensitivity to initial conditions and deterministic pseudorandomness, enhances encryption security. These characteristics make the CLM suitable for generating cryptographic keys and enhancing the security of encryption algorithms. It is defined by the following:

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

where x is a value in the interval $(0, 1)$, and r is a control parameter. The system exhibits chaotic behavior when r is set to 3.57 or higher. This characteristic enables it to generate robust cryptographic keys and enhance pixel substitution [7,21]. For instance, Figure 2 illustrates the bifurcation diagram of the logistic map, characterized by the initial condition $x_0 \in [0, 1]$ and parametrized by the control parameter $r \in [0, 4]$. Within the range of $[3.57, 4]$, the CLM exhibits chaotic behavior, where subtle variations in the initial condition lead to pronounced oscillations in the resulting values. Consequently, the system becomes aperiodic and divergent.

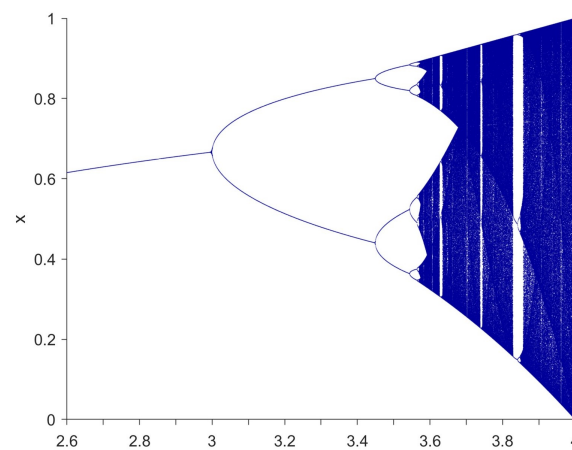


Figure 2. Bifurcation diagram of the logistic map.

2.1.2. The Substitution–Permutation Network (SPN)

The SPN architecture, as proposed by Shannon in 1949, serves as a cornerstone of contemporary encryption algorithms [8]. It entails the iterative application of substitution operations using an S-box and permutation operations using a P-box to obfuscate the correlation between the plaintext and the ciphertext [22]. An SPN framework comprises multiple rounds, each encompassing a substitution phase in which the plaintext is transformed via an S-box, followed by a permutation phase in which the bits are rearranged using a P-box. This process can be mathematically represented as follows:

$$y = P(S(x)), \quad (2)$$

where S is the substitution function, P is the permutation function, x is the input plaintext, and y is the output ciphertext.

However, the security of an SPN-based encryption scheme hinges on the quality of the S-box and P-box designs. A strong S-box should demonstrate non-linearity and resistance to differential and linear cryptanalysis. At the same time, the permutation step ensures diffusion by spreading the influence of plaintext bits throughout the ciphertext [7].

2.1.3. Row and Column Permutation

A permutation based diffusion mechanism is employed for row and column shuffling. It shuffles pixel positions to obscure spatial correlations and modifies pixel values to

enhance security. Using a chaotic sequence further ensures sensitivity to initial conditions, rendering the encryption highly resistant to minor variations in input keys. This enhances the overall security of the encryption, as the shuffled data becomes significantly different from the original data despite the rearranged pixel values remaining the same.

2.1.4. The Encryption and Decryption Phases

Figure 3 presents the flowchart for the encryption phase. The flowchart illustrates that the encryption scheme employs a combination of cryptographic techniques, specifically the CLM and SPN, to implement lightweight encryption that operates efficiently under resource constrained conditions. In this way, the proposed algorithm offers an added advantage by enhancing the diffusion and confusion properties of the encryption without significantly compromising computational efficiency. Algorithm 1 is adapted from our previously published work titled ‘Z-Crypt: Chirp Z-Transform-Based Image Encryption Leveraging Chaotic Logistic Maps and Substitution Permutation Network’ [23].

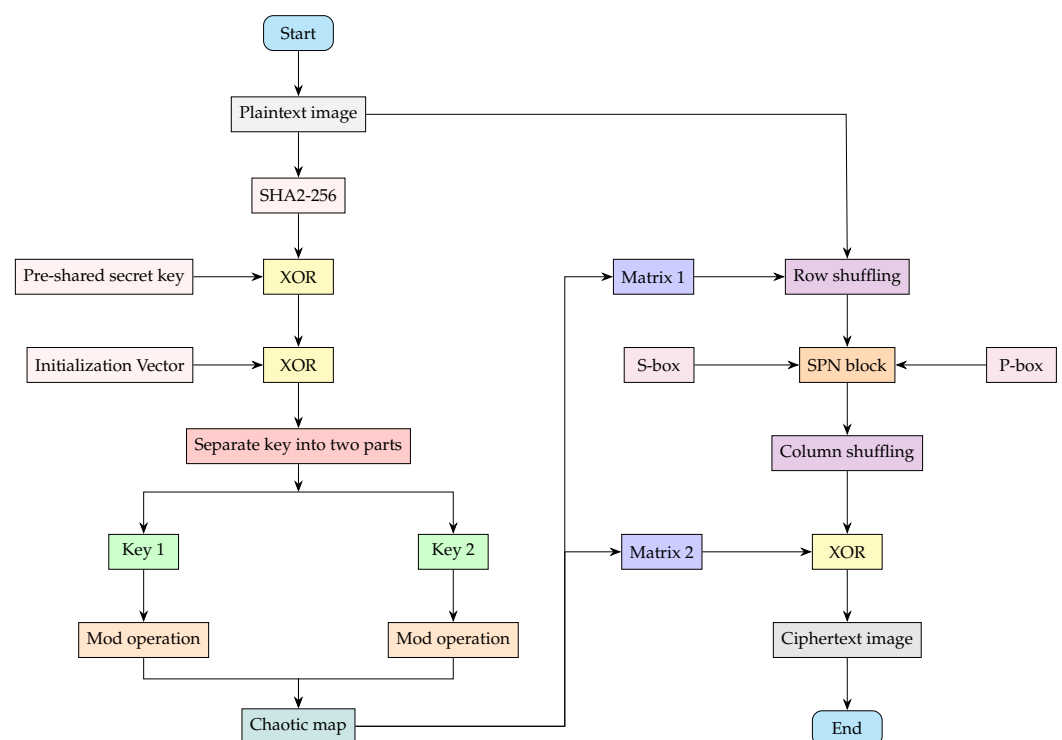


Figure 3. Flowchart of the image encryption phase.

The SHA2-256 hash is applied to the plaintext message at the beginning of the encryption process. The result is a fixed length 256-bit hash, irrespective of the plaintext size (lines 1–2). The goal of hashing the plaintext is to transform the input into a unique and secure form. Even slight modifications of the plaintext will produce a very different hash, which enhances the integrity of the encryption scheme.

Subsequently, the result of the SHA2-256 hash is then XORed with the pre-shared secret key (Pk), resulting in a derived key. The pre-shared secret key is an important component of the encryption process, as its primary purpose is to ensure that decryption is impossible without it. After this, a 256-bit key called the initialization vector (IV) is randomly generated and XORed with the derived key to give a unique key. Then, the result of the XOR operation is divided into two parts to create two different keys, each 128 bits long. These two 128-bit parts undergo a modular arithmetic operation to produce key_1 and key_2 , which are used later in the encryption process. The mod (0.9999) operation is used to

limit the values of the keys between 0 and 1 to prepare them for use in the chaotic map. This process occurs in steps 3 to 8 of Algorithm 1.

Since there is a need to generate different matrices for different parts of the encryption process, the CLM is chosen because of its ability to generate pseudorandom numbers that are highly sensitive to initial conditions. The initial value of the chaotic map (x_0) is set to the value of key_1 as it serves as the seed value. The control parameter (r) is set to 3.99876 as the logistic map has become highly chaotic at that point. A continuous iteration of the chaotic map is then performed to generate the chaotic sequences Matrix 1 and Matrix 2.

Algorithm 1 Proposed Image Encryption Algorithm

Input: Plaintext image P , pre-shared key Pk , Initialization Vector IV
Output: Cipher image C , Hash H

```

1:  $[H, W, nc] \leftarrow \text{reshape } 2D(P)$  // Reshape image into  $H \times W \times nc$ , where  $nc$ 
   is number of channels
2:  $H \leftarrow \text{hash}(P)$ 
3:  $Key \leftarrow H \oplus Pk$ 
4:  $Key \leftarrow Key \oplus IV$ 
5:  $halfLen \leftarrow \text{length}(Key)/2$ 
6:  $spnKey \leftarrow Key(1 : halfLen)$ 
7:  $Key1 \leftarrow Key(1 : halfLen) \bmod m$  // where  $m$  is 0.9999
8:  $Key2 \leftarrow Key(halfLen : \text{end}) \bmod m$ 
9: for  $n \leftarrow 1$  to  $H$  do
10:    $x1(n+1) \leftarrow r \times x1(n) \times (1 - x1(n))$  // where  $x1(1)$  is  $Key1$ ,  $r$  is 3.99879
11:    $M1(n) \leftarrow x1(n+1) \times 10^6 \bmod H$ 
12: end for
13: for  $n \leftarrow H+1$  to  $H+W$  do // For width
14:    $x1(n+1) \leftarrow r \times x1(n) \times (1 - x1(n))$ 
15:    $M1(n) \leftarrow x1(n+1) \times 10^6 \bmod H$ 
16: end for
17: for  $n \leftarrow 1$  to  $H \times W$  do
18:    $x2(n+1) \leftarrow r \times x2(n) \times (1 - x2(n))$  // where  $x2(1)$  is  $Key2$ 
19:    $M2(n) \leftarrow x2(n+1) \times 10^6 \bmod 256$ 
20: end for
21:  $Encrypt \leftarrow \text{rowShuffle}(P, M1(1 : H))$  // in Algorithm 2
22:  $Encrypt \leftarrow \text{runSPN}(Encrypt, spnKey)$  // in Algorithm 3
23:  $Encrypt \leftarrow \text{columnShuffle}(Encrypt, M1(H+1 : H+W))$  // in Algorithm 4
24:  $C \leftarrow Encrypt \oplus M2$ 
25: return  $C, H$ 
26: end
  
```

Matrix 1 is generated using one instance of CLM iteration with the initial value of key_1 . The number of elements to be generated is given by the following:

$$M_1 = N_{(\text{row})} + N_{(\text{col})}, \quad (3)$$

where $M1$ represents the number of elements in this sequence, N_{row} represents the number of rows in the plaintext, and N_{col} is the number of columns of the plaintext.

A modulus operation is applied to the values after separating the two arrays. This will effectively adjust the values of these arrays within the necessary range. Since it is expected that the number of rows may not be equal to the number of columns, the modulus operation is performed separately for N_{row} elements and N_{col} elements. This is followed by a floor operation to ensure that all values are integers. The row and column elements are concatenated together to make Matrix 1. The equation employed in this operation is defined as follows:

$$\begin{aligned}
 M_{(\text{row})} &= \left\lfloor x_r * 10^6 \bmod N_{(\text{row})} \right\rfloor \\
 M_{(\text{col})} &= \left\lfloor x_c * 10^6 \bmod N_{(\text{col})} \right\rfloor \\
 M1 &= [M_{(\text{row})} \parallel M_{(\text{col})}],
 \end{aligned} \tag{4}$$

where M_{row} and M_{col} are the row and column element arrays, x_r is the portion of the chaotic sequence dedicated for the row element matrix, and x_c is that of the column elements matrix. Matrix 2 is generated using another instance of the CLM. key_2 is used as the initial value. Matrix 2 contains $(N_{\text{row}} * N_{\text{col}})$ elements. This makes it the same size as the plaintext data. After generating the sequence data, the modulus and floor operations are performed as defined:

$$M2 = \left\lfloor x_{rc} * 10^6 \bmod 256 \right\rfloor, \tag{5}$$

where $M2$ represents Matrix 2 and (x_{rc}) is the sequence generated, containing the same number of elements as the plaintext. The data is then reshaped to the same size as the plaintext. The key generation processes are performed in steps 9 to 11 of Algorithm 1.

The row permutation operation is the first transformation operation performed on the data. The $M_{(\text{row})}$ array saved in the $M1$ matrix is used as the key for this operation. The rows are shuffled based on the index values stored in $M_{(\text{row})}$, with each element in $M_{(\text{row})}$ corresponding to a target row position. This means that each row in the plaintext data is swapped with another row determined by the value of $M_{(\text{row})}$. This is presented in step 12 of Algorithm 1 and the pseudocode is described in Algorithm 2. This process ensures that the spatial structure of the data is scrambled in a pseudorandom manner driven by the key generated from the $M_{(\text{row})}$. The operation is described as follows:

$$C_{M_{(\text{row})}}^{\text{rp}}(i, j) = P_{(i, j)}, \quad \text{for } \left\{ \begin{array}{l} i = 1, 2, \dots, N_{(\text{row})} \\ j = 1, 2, \dots, N_{(\text{col})} \end{array} \right\}, \tag{6}$$

where i and j are the row and column indices respectively used to index the pixels. The substitution–permutation network (SPN) is then applied to further obfuscate the pixel values to provide an additional encryption layer to enhance security, as indicated in step 13 of Algorithm 1. This transformation consists of two main components: the S-box and the P-box. In this respect, the S-box is in charge of substituting every image pixel with another value according to a predefined mapping table; the mapping is generated using key_1 . The non-linearity introduced by the substitution makes the relationship between the original and encrypted pixel values highly complex and difficult to predict. After the substitution operation, the substituted pixel values are rearranged using the P-box, based on key_2 , which spreads the pixel values across the entire image and enhances diffusion.

Algorithm 2 Row Shuffling Procedure

Input: Plaintext image P , Chaos sequence $M1_{\text{rows}}$

Output: Row shuffled image P_{row}

```

1:  $[H, W] \leftarrow \text{size}(P)$  // Image size is  $H \times W$ 
2: for  $i \leftarrow 1$  to  $H$  do
3:    $\text{targetIdx} \leftarrow \text{floor}(M1_{\text{rows}}(i)) \bmod H + 1$ 
4:   Swap rows  $P(i, :)$  and  $P(\text{targetIdx}, :)$ 
5: end for
6: return  $P_{\text{row}} \leftarrow P$ 
7: end

```

To further increase the complexity, the arrays in both the S-box and P-box are generated using the CLM, which makes such operations highly sensitive to the initial conditions

and keys. The CLM is also utilized to generate a matrix used in the XOR operation in the SPN. The XOR operation mixes the substitution and permutation values with the chaotic sequence, ensuring that the final encrypted output is highly resistant to different cryptographic attacks, thus increasing the unpredictability of the encryption process. This process is described in step 15 of Algorithm 1. The SPN operation is applied to the cipher image (C) generated by the row permutation operation using the following equation:

$$C_{(SPN)} = P(S(C_{(i,j)}, k1), k2) \oplus H_{(i,j)}, \quad (7)$$

where $P(\cdot)$ is the P-box operation, $S(\cdot)$ is the S-box operation, and $C_{(i,j)}$ represents the cipher data point at coordinate i, j . Similarly, $k1$ and $k2$ are the keys for the S-box and P-box operations and $H_{(i,j)}$ is the matrix generated for the XOR operation. Algorithm 3 describes the process for the SPN transformation.

Algorithm 3 Substitution–Permutation Network (SPN)

Input: Shuffled image P_{row} , SPN key ($spnKey$)
Output: Transformed image P_{spn}
 1: $[k_1, k_2] \leftarrow \text{split}(spnKey)$
 2: $S \leftarrow \text{Apply S-Box}(P_{row}, k_1)$ // Substitution Box
 3: $P_{spn} \leftarrow \text{Apply P-Box}(S, k_2)$ // Permutation Box
 4: **return** P_{spn}
 5: **end**

The column permutation operation is then performed using the $M_{(col)}$ array generated as part of the matrix $M1$. The column permutation operation is similar to the row permutation, except that the columns are being shuffled as described in Algorithm 4. These continuous operations of substitution and permutation assist in creating confusion and diffusion in the encryption process. The column operation is expressed as follows:

$$C_{M_{(col)}}^{cp}(i,j) = C_{(i,j)}, \quad (8)$$

As a final step, the resulting cipher data generated from the column operation is then XORed with the previously generated $M2$ matrix. This further obscures the cipher, making it more complex and resistant to cryptanalytic attacks. The performed XOR operation is expressed as follows:

$$C_{M_{(col)}}^{cp}(i,j) = C_{(i,j)} \quad C = C_{(i,j)} \oplus M2_{(i,j)}, \quad (9)$$

where $C_{(i,j)}$ is the cipher obtained from the column permutation operation, and $M2_{(i,j)}$ represents the pseudorandom sequence Matrix 2 generated using the CLM.

Algorithm 4 Column Shuffling Procedure

Input: Transformed image P_{spn} , Chaos sequence $M1_{cols}$
Output: Column shuffled image P_{col}
 1: $[H, W] \leftarrow \text{size}(P_{spn})$
 2: **for** $j \leftarrow 1$ to W **do**
 3: $targetIdx \leftarrow \text{floor}(M1_{cols}(j)) \pmod{W} + 1$
 4: Swap columns $P_{spn}(:, j)$ and $P_{spn}(:, targetIdx)$
 5: **end for**
 6: **return** $P_{col} \leftarrow P_{spn}$
 7: **end**

2.1.5. Steps to Perform the Decryption Process

The decryption process follows the reverse order of encryption steps and can be summarized as follows:

1. First, retrieve the cipher image C , initialization vector (IV), and SHA2-256 hash.
2. Recompute the chaotic matrices $M1$ and $M2$ by performing steps similar to encryption.
3. Perform an XOR operation between the ciphertext C and the matrix $M2$.
4. Apply an inverse column shuffle using M_{col} from $M1$.
5. Reverse the SPN transformation, first applying the inverse P-box operation and then the inverse S-box operation.
6. Apply an inverse row shuffle using M_{row} from $M1$.
7. Recover the original plaintext image.

2.2. Data Embedding and Extraction Process

After encryption, the ciphertext is embedded into a cover image to maintain secrecy while minimizing perceptual distortion. This process ensures the carrier image retains its visual quality while enhancing security.

2.2.1. Steganography with Least Significant Bit (LSB)

Steganography hides secret data within a cover image to avoid detection. A widely used approach is LSB substitution, where the LSBs of pixel values are modified to embed cryptographic information without noticeable alterations to the image [24]. This technique ensures efficient data embedding with minimal impact on pixel intensity. For an 8-bit grayscale image, the LSB substitution modifies pixel value $I(i, j)$ as follows:

$$I^l(i, j) = (I(i, j) \text{ AND } 0xFE) \text{ OR } K(i, j), \quad (10)$$

where $I^l(i, j)$ is the modified pixel value, $I(i, j)$ is the original pixel value, and $K(i, j)$ is the secret data bit. The AND operation with $0xFE$ (binary [11111110]) clears the LSB, while the OR operation with $K(i, j)$ sets it according to the secret data.

The LSB substitution process first converts the secret data into a binary sequence. Each bit of the secret data is embedded one at a time into the LSB of the pixel values. If the bit of the secret data is 0, the AND operation ensures the LSB remains 0; if the bit is 1, the OR operation modifies it to 1. This method achieves lightweight and efficient data embedding while preserving the original image's appearance.

2.2.2. The Steganography Embedding Phase

Figure 4 presents the flowchart of the steganography embedding phase, which consists of three steps: header generation, CLM shuffling, and data bit embedding. Since digital data (including images, audio, videos, and text) is encoded in binary, the embedding process involves converting the input data into a bitstream. Algorithm 5 presents the pseudocode for the proposed steganography embedding phase.

Three modes of data embedding are considered: alphanumeric, image, and file steganography. Alphanumeric embedding hides textual data, image embedding inserts an image into the cover image, and file embedding converts an entire file into binary before embedding it, irrespective of its content. To facilitate accurate data extraction, a header containing metadata such as size, type, and file name is added at the start of the hidden data. The header size depends on the embedding mode and ensures secure and efficient steganography.

Before embedding, the size of the cover image is stored, as it can be reshaped during processing. If the cover image is not grayscale, it is converted into a 2D matrix, while the secret data is transformed into a 1D binary array. Note that the algorithm only assumes the cover image can be converted into a 2D matrix of 8 bits and converted back into its original format. This holds true for two, three, and four channel images. Thus, the cover image is read and available as a matrix of uint8 data, which can be reshaped as necessary.

The header and secret data are then merged into a binary bitstream since LSB substitution operates in binary by modifying the LSBs of the pixel values. These processes are covered in the first four steps of Algorithm 5.

Algorithm 5 Proposed Steganography Embedding Phase

Input: Data, Mode, Cover Image, Filename
Output: Stego-image

```

// Get header information
// Rows and columns are in 16 bits
1:  $rows \leftarrow [rows\_lsb, rows\_msb]$ 
2:  $columns \leftarrow [columns\_lsb, columns\_msb]$ 
3:  $filename \leftarrow \text{native2bin}(filename)$ 
4:  $header \leftarrow [mode, rows, columns, filename]$ 
5:  $D \leftarrow \text{reshape}(data, 1, [])$ 
   // Reshape to 1D
6:  $D_b \leftarrow \text{getBitstream}(D)$ 
7:  $D_h \leftarrow [header, D_b]$ 
   // Compute indexes data using CLM
8: for  $n \leftarrow 1$  to  $H + W$  do
9:    $x_{clm}(n+1) \leftarrow r \times x_{clm}(n) \times (1 - x_{clm}(n))$ 
   //  $x_{clm}(1) = \text{Key}_1, r = 3.99879$ 
10: end for
11:  $rowCLM \leftarrow x_{clm}(1 : H)$ 
12:  $colCLM \leftarrow x_{clm}(H + 1 : H + W)$ 
   // Shuffle the bits of cover image
13:  $X \leftarrow \text{reshape}(cover, 1, [])$ 
   // Perform X using rowCLM and colCLM as index
14:  $X_r \leftarrow X(rowCLM, colCLM)$ 
   // Select n pixels corresponding to the number of bits in the
   augmented data
15:  $X_s \leftarrow X_r(1 : \text{length}(D_h))$ 
   // Perform AND operation as in Equation (11).
16:  $X_c \leftarrow X_s \& (X_s \oplus (2^b - 1))$ 
   // Perform OR operation as in Equation (12).
17:  $X_e \leftarrow X_c | D_h$ 
18:  $X_r(1 : \text{length}(D_h)) \leftarrow X_e$ 
   // Reverse the shuffle operation
19:  $X_r \leftarrow X_r(rowCLM_{inv}, colCLM_{inv})$ 
   // Reshape to stego-image
20:  $X_{stego} \leftarrow \text{reshape}(X_r, H, W)$ 
21: return  $X_{stego}$ 

```

To enhance security, two matrices, Matrix 1 and Matrix 2, are generated using the CLM based on the number of rows and columns in the cover image. Matrix 1 corresponds to the rows, while Matrix 2 corresponds to the columns. These matrices introduce randomness, ensuring unpredictability in data positioning. The cover image pixels are shuffled using these matrices, disrupting the natural order and making detection more difficult. The shuffled image is then converted into a one dimensional array for uniform processing.

Pixels corresponding to the number of hidden data bits are selected sequentially to serve as carriers for LSB substitution. As the LSB method modifies only the LSBs, changes in pixel intensity remain imperceptible to the human eye. Matrix operations optimize the embedding process. In a single bit operation, the LSB of a pixel is replaced by a data bit. To embed multiple bits per pixel, a bitwise AND operation clears the required least significant b bits before substitution:

$$P_{(\text{new})} = P_{(\text{orig})} \& \left(P_{(\text{orig})} \oplus (2^b - 1) \right), \quad (11)$$

where P_{new} is the modified pixel value, P_{orig} is the original pixel value, and b is the number of bits to be cleared. The XOR operation ensures opposite values at the target bits, while the AND operation sets the least significant b bits to zero.

Next, a bitwise OR operation embeds the data bits into the prepared pixels. If b is greater than 1, the data is resized into n -bit values to allow a single matrix operation, improving efficiency:

$$P_{(\text{emb})} = P_{(\text{new})} | P_{(\text{res})}, \quad (12)$$

Finally, the selected pixels in the shuffled image are replaced with the embedded pixels. An inverse row and column shuffle restores the original image structure, followed by reshaping if necessary. The resulting steganographic image retains its visual integrity while securely embedding the hidden data. The pseudocode for the steganography embedding phase is presented in Algorithm 5.

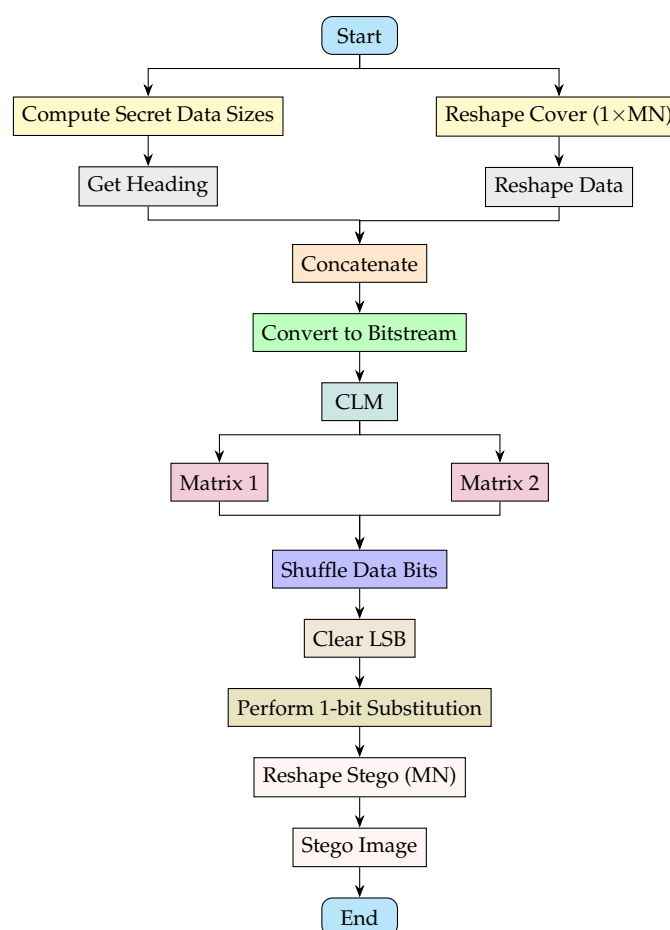


Figure 4. Flowchart of the steganography embedding phase for 1-bit substitution.

2.2.3. Steganography Extraction Process

The steganography extraction process involves reversing the embedding procedure described in Algorithm 5. The following steps are performed:

1. Apply steps 6–8 from the embedding process using the steganographic image.
2. Reshape the steganographic image into a one dimensional (1D) array.
3. Extract the first 560 elements of the reshaped array, ensuring the header is captured within this range. This is to account for longer file names when the secret data is a file.

4. Perform a bitwise AND operation between the selected data and 1 to extract the LSBs of the pixels.
5. Reshape the resulting data into an $(8 \times N)$ matrix and convert it to uint8.
6. Decode the header information to determine the embedding mode, row size, and column size, and check if a file name is provided, indicating whether the extracted data should be saved to the disk.
7. Compute the header size based on the data required for storing all metadata information.
8. Determine the size of the hidden data using the extracted row and column sizes.
9. Extract the data pixels using the computed size multiplied by 8 since each pixel contains a single bit of data.
10. Perform a bitwise AND operation between the extracted data and 1 to retrieve the LSBs of all pixels.
11. Reshape the result into an $(8 \times N)$ matrix and convert it to uint8, yielding the final extracted data.

2.2.4. Combining Encryption and Embedding

The secret data is first loaded, and the total number of elements is evaluated. If the data lacks a 2D structure and its size exceeds 100 bytes, it is reshaped into a 2D format. The reason for the 100 bytes is that all data in this study is loaded as a group of 8 bits. This restructuring is achieved by adding minimal padding to the data vector, ensuring that it forms a balanced matrix shape. An additional arbitrary character value (set to 3 as ASCII end-of-text code) is used for padding to maintain data consistency, enabling efficient 2D encryption. This is only used when the secret data lacks a 2D structure and there is an odd number of bytes within.

Next, the reshaped data undergoes encryption, producing a cipher image along with key data containing the pre-shared key and an initialization vector (IV). The encryption ensures confidentiality by preventing unauthorized access or interpretation of the original data. Based on the selected mode, the ciphered data may be further reshaped to facilitate seamless embedding.

The encrypted data is then embedded into a cover image using the LSB technique. This process requires the ciphertext and cover images, with an optional file name for accurate file recovery during extraction. The LSB method ensures that the encrypted data remains imperceptible to the human eye by modifying only the LSBs of the cover image pixels. This preserves the cover image's appearance while securely concealing the data, establishing a robust steganographic layer for secure data transfer and storage.

3. Experimental Results and Performance Analysis

This section presents various experiments conducted to evaluate the performance of the proposed algorithm. Several performance metrics are considered, including the visual quality, embedding capacity, mean square error (MSE), and PSNR. The performance is evaluated on a Windows 10 Pro system equipped with an Intel® Xeon® CPU E5-1620 v4 running at 3.50 GHz and featuring 32.0 GB of RAM. We use MATLAB R2023b to run the proposed algorithm, performing all tests 20 times.

A selection of cover images with dimensions of 256×256 , 512×512 , and 1024×1024 pixels are used for testing in both grayscale (8-bit) and color (24-bit) formats. These images were sourced from the standard USC Image Database [25], as illustrated in Figure 5. Due to the nature of the LSB substitution method, there are multiple ways to compute the maximum number of embedding bits that can be inserted. These ways depend on the number of LSBs (k -LSB) substituted per pixel. The properties of these images and their maximum 1-bit capacities are summarized in Table 1. We note that the payload for 1-LSB is

considered to be 1 bpp. This is one bit per pixel, such that k -LSB is then k bpp, which thus means k bits per pixel.

Table 1 shows that the Baboon image has a capacity of 786,432 bits, which is less than 100 KB of data. When the Clock image is loaded as an image, it contains 524,288 bits of data. This indicates that the Clock image can be successfully embedded inside the Baboon image with substantial remaining capacity. However, a data compression approach can also be used to handle the payload by reducing its size and reading the data as a binary file in uint8 format, thereby increasing the effective embedding capacity.



Figure 5. The cover images: (a) Clock, (b) Sailboat, (c) Male, (d) Tree, (e) Baboon, (f) Boat, (g) Peppers, and (h) Cameraman.

Table 1. Properties of the images used in the evaluations.

Image	Image Size	1-LSB Capacity (Bits)	Type
Clock	256 × 256	65,536	Gray
Sailboat	512 × 512	786,432	Color
Male	1024 × 1024	1,048,576	Gray
Tree	256 × 256	196,608	Color
Baboon	512 × 512	786,432	Color
Boat	512 × 512	262,144	Gray
Peppers	512 × 512	786,432	Color
Cameraman	512 × 512	262,144	Gray

3.1. Execution Time

The embedding process involves converting the secret data into a binary stream, modifying pixels using the LSB substitution method, and reshuffling the pixels using the CLM. The extraction process reverses these operations, retrieving the hidden data while maintaining the integrity of the cover image.

Table 2 presents the execution time for embedding the ASCII characters of "Hello Steganography" in different plaintext images. The execution time results are averaged over 20 iterations, and the final result is reported. To evaluate the algorithm's performance for larger data sizes, Table 3 reports the execution time when embedding a 10 KB randomly generated file containing 10,240 alphanumeric characters into the cover images.

The results indicate that embedding time varies depending on the image content and complexity. Higher embedding times are observed for more detailed images due to pixel

variations affecting LSB substitution and chaotic shuffling. The extraction process, however, is consistently faster since it only involves reversing the pixel modifications and retrieving the hidden bits. The average total execution time across all tested images is 0.0113 seconds. Also, the average total execution time when embedding 10 KB of data is 0.0162 seconds. These results demonstrate the efficiency of the proposed algorithm.

Table 2. Execution time and extraction time (in seconds) for embedding “Hello Steganography” in different images.

Cover Image	Embedding Time (s)	Extraction Time (s)	Total Time (s)
Clock	0.0042	0.0025	0.0067
Sailboat	0.0094	0.0032	0.0126
Male	0.0104	0.0033	0.0136
Tree	0.0047	0.0026	0.0073
Baboon	0.0119	0.0031	0.0150
Boat	0.0084	0.0029	0.0113
Peppers	0.0123	0.0031	0.0154
Cameraman	0.0048	0.0027	0.0075
Average	0.0083	0.0030	0.0113

Table 3. Execution time and extraction time (in seconds) for embedding 10 KB of data in different plaintext images.

Cover Image	Embedding Time (s)	Extraction Time (s)	Total Time (s)
Clock	0.0070	0.0059	0.0129
Sailboat	0.0110	0.0054	0.0164
Male	0.0136	0.0063	0.0199
Tree	0.0071	0.0053	0.0124
Baboon	0.0144	0.0060	0.0204
Boat	0.0101	0.0055	0.0155
Peppers	0.0139	0.0057	0.0196
Cameraman	0.0077	0.0051	0.0128
Average	0.0106	0.0057	0.0162

From Tables 2 and 3, it can be observed that the execution time for embedding is longer than for extraction. This is expected because the embedding process involves more data processing steps than the extraction time. The execution times in Table 3 are significantly larger than those in Table 2, primarily due to the difference in data size. A comparison of the execution time with other methods from the literature is shown in Table 4. This runtime is computed based on the average runtime of embedding 10 KB in all of the test images.

Table 4. The execution time (in seconds) of the proposed algorithm for embedding 100 KB of data compared with other methods in the literature.

Methods	Embedding Time (s)	Extraction Time (s)	Total Time (s)
[26]	2.50	2.00	4.50
[27]	0.83	1.50	2.33
Proposed	0.103	0.032	0.135

Table 4 shows that the proposed algorithm outperforms other methods from the literature. This is due to the absence of long for loops in the embedding process. All computations are performed using bit manipulation and vector operation strategies. Also, larger data is converted to two dimensions in the encryption process to leverage the advantage of the fast, efficient, and secure image encryption method employed. Table 5 presents

the execution time when embedding the Clock and Tree images as secret data into the different cover images. To perform this experiment, there is a need to embed in more space than the number of pixels available in a cover image. Thus, k-LSB is utilized such that, when data cannot be embedded further in a significant bit, a higher significant bit is selected for the embedding. However, since the steganography images degrade faster at higher LSBs, the limit is set to 4-LSB.

Table 5. Execution time (in seconds) for embedding Clock and Tree secret images into different cover images.

Cover Image	Embedding Time (s)		Extraction Time (s)	
	Clock	Tree	Clock	Tree
Clock	–	–	–	–
Sailboat	0.0379	0.0681	0.0376	0.0824
Male	0.0454	0.0691	0.0413	0.0798
Tree	0.0390	–	0.0444	–
Baboon	0.0374	0.0907	0.0342	0.1080
Boat	0.0389	–	0.0427	–
Peppers	0.0376	0.0948	0.0352	0.1131
Cameraman	0.0375	–	0.0428	–

From Table 5, it can be observed that some values are unavailable. This is because the available embedding capacity of the cover image is insufficient. It is noted that a limit of 4-LSB is set for this experiment. This means that, if the size of the secret data to be embedded is greater than 4 bits, it is considered an invalid operation and skipped. Figure 6 presents the resulting steganographic images after the Clock and Tree images have been embedded.

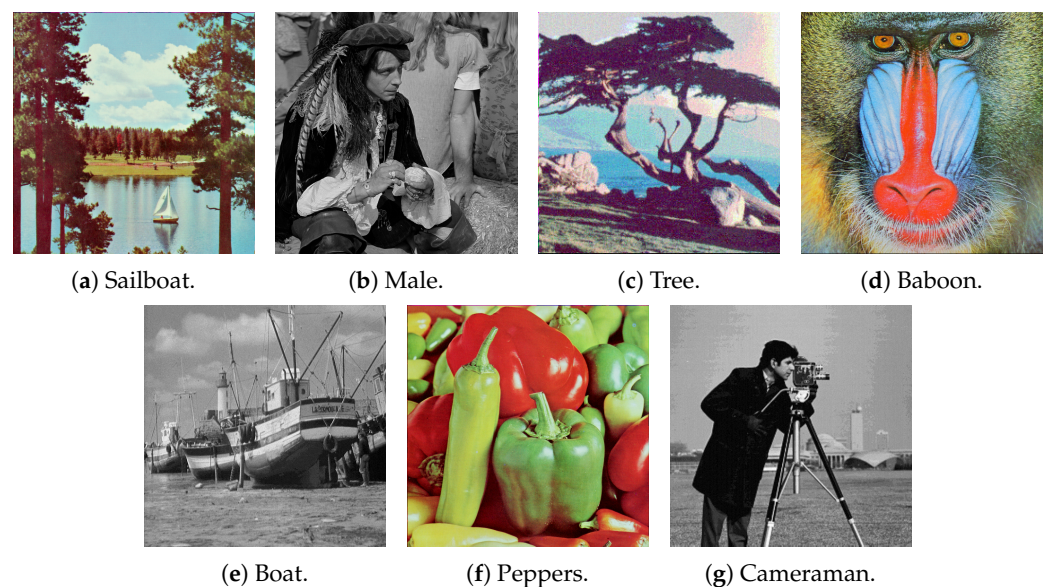


Figure 6. The resulting steganographic images after embedding the Clock secret image: (a) Sailboat, (b) Male, (c) Tree, (d) Baboon, (e) Boat, (f) Peppers, and (g) Cameraman.

From Figure 6, it can be seen that there is very little distortion in the steganographic images. This shows that the proposed algorithm is effective enough to avoid human visual perception. However, the Clock image is also not present in the result. This is because the Clock cover image is unable to contain the Clock secret image. This is expected, since eight times the number of pixels would be required, which is much larger than the

predefined maximum of four times. Figure 7 shows the result of embedding the secret Tree image within the various cover images.

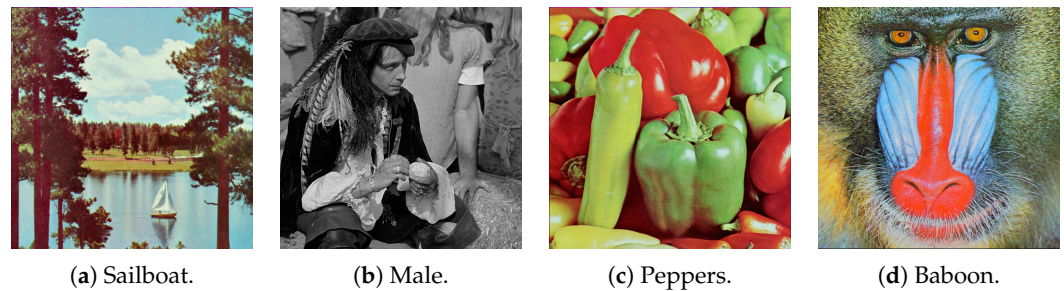


Figure 7. Steganographic images after embedding the Tree secret image using our proposed method: (a) Sailboat, (b) Male, (c) Peppers, and (d) Baboon.

Figure 7 shows that slight distortion can still be noticed after embedding the Tree image. This suggests that embedding the Tree image does not significantly impact human visual perception of the four images. However, this is not the case for the Tree, Boat, and Cameraman, as the number of bits to be embedded surpasses the required maximum of 4-LSB previously set.

3.2. Mean Square Error (MSE) Analysis

The MSE is a metric widely used in image processing, cryptography, and steganography to quantify the difference between an original and a modified image. It computes the average squared difference between corresponding pixel values, effectively penalizing larger discrepancies more than smaller ones. This characteristic makes the MSE advantageous in some applications while limiting in others. A key benefit of the MSE is its computational efficiency, allowing for rapid image quality assessment.

The MSE between the cover and its corresponding steganographic images is given by the following:

$$\text{MSE} = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W [C(i, j) - S(i, j)]^2, \quad (13)$$

where H and W denote the image height and width, respectively, while $C(i, j)$ and $S(i, j)$ represent the pixel values of the cover and steganographic images at position (i, j) .

Table 6 presents the computed MSE values for various plaintext images when embedding 19 bytes (19 B) and 10 KB of data. Additionally, Table 7 compares the MSE values of the proposed method against those reported by Emam et al. [16], highlighting the method's effectiveness.

The results indicate that the proposed method consistently achieves lower MSE values than the method presented by Emam et al. [16], demonstrating its superior ability to preserve image quality while embedding secret data. This improvement suggests that the proposed technique effectively maintains imperceptibility, a critical requirement for secure steganographic applications.

Table 6. MSE values for steganographic images with embedded data sizes of 19 B and 10 KB.

Image	MSE (19 B)	MSE (10 KB)
Clock	0.0001	0.0152
Sailboat	0.0001	0.0263
Male	0.0002	0.0785
Tree	0.0001	0.0257
Baboon	0.0001	0.0258
Boat	0.0001	0.0754
Peppers	0.0001	0.0272
Cameraman	0.0001	0.0205
Average	0.0001	0.0368

Table 7. Comparison of MSE values of the proposed method and other methods.

Image	Capacity (Bytes)	Emam et al. [16]	Proposed
Baboon	43,690	0.4253	0.0138
	32,768	0.3184	0.0103
	21,845	0.2120	0.0070
Peppers	43,690	0.4243	0.1162
	32,768	0.3178	0.0873
	21,845	0.2111	0.0581

3.3. Peak Signal-to-Noise Ratio (PSNR) Analysis

The PSNR is a metric widely used for evaluating image quality by measuring the ratio between the maximum possible signal power and the noise introduced by image modifications [28]. The PSNR is derived from the MSE between the original and modified images, where higher PSNR values indicate a lower perceptible distortion. The PSNR is computed as follows:

$$\text{PSNR} = 10 \log_{10} \left(\frac{\text{MAX}_p^2}{\text{MSE}} \right), \quad (14)$$

where MAX_p^2 represents the squared maximum possible pixel value, typically 255^2 for 8-bit images.

Table 8 presents the PSNR results for various cover images when embedding 19 B and 10 KB of data. The results indicate that embedding a smaller payload (19 B) results in significantly higher PSNR values, implying minimal perceptible changes to the cover image.

Table 8. PSNR values for steganographic images with different embedding capacities.

Image	PSNR (19 B)	PSNR (10 KB)
Clock	85.503	59.185
Sailboat	87.356	60.957
Male	82.493	56.183
Tree	87.449	60.956
Baboon	86.875	60.979
Boat	82.822	56.175
Peppers	87.219	60.956
Cameraman	86.188	59.176
Average	85.738	59.320

The observed trends indicate that lower embedding rates result in better image quality. For example, the PSNR for the Baboon image is 86.875 with a 19 B payload but drops to 60.979 when embedding 10 KB. Similar patterns are evident in all the images tested.

Table 9 compares the PSNR values of the proposed method with those reported in previous studies, illustrating its superior performance. The results indicate that the proposed method achieves a higher PSNR compared to previous techniques, reflecting its ability to embed data while maintaining superior image quality. Tables 10 and 11 further validate its efficiency by comparing the performance with Wu and Tsai [29] and Hameed et al. [26].

Table 9. PSNR comparison with existing methods.

Image	Capacity (Bits)	[16]	[30]	Proposed
Baboon	144916	55.68	38.44	58.51
Peppers	145995	55.67	42.28	58.46
Sailboat	143278	–	40.66	58.45

Table 10. Comparison of PSNR with Wu and Tsai.

Image	Capacity (Bits)	[29]	Proposed
Baboon	56,291	37.90	62.58
Peppers	50,685	41.73	63.07

Table 11. Comparison of PSNR with Hameed et al.

Image	[26]		Proposed	
	Capacity (Bits)	PSNR	Capacity (Bits)	PSNR
Baboon	79,848	36.01	80,000	61.06
Peppers	74,987	39.89	80,000	61.35
Boat	76,084	38.34	80,000	56.48

The findings indicate that the proposed method achieves notable improvements in PSNR, confirming its effectiveness in preserving image quality while embedding data. Even when compared to existing methods, the approach maintains higher imperceptibility and lower distortion, making it a promising choice for secure image steganography.

3.4. Pixel Difference Histogram Security Analysis

The pixel difference histogram is a steganalysis method that examines the distribution of pixel intensity differences to detect hidden information in images. This method operates by computing the differences between adjacent pixels, which, in natural images, are expected to follow specific statistical patterns [26]. When secret data is embedded using methods such as LSB steganography, these intensity differences often appear more random and deviate from their typical distribution. Plotting a histogram of these pixel differences allows forensic analysts to identify anomalies that can point to the existence of concealed data [31].

The main goal of the pixel difference histogram analysis is to look at how regular and smooth the pixel variations are. For example, pixel differences in natural images typically follow a predictable pattern, with smaller variations between neighboring pixels occurring more frequently. However, secret data is introduced when steganography is used, changing this pattern and causing odd spikes or histogram shifts. By showing how much the pixel difference values have changed, the analysis can also be used to distinguish between different steganographic methods. This technique is helpful since it is reasonably

easy to use and efficient at identifying even minute alterations in the pixel level structure of an image [32].

The Baboon and Peppers images are used as test images for the pixel difference histogram (PDH). Three different tests are performed based on the data capacity to be embedded. These are 10 KB, half the complete image size, and the full image size of the cover. It should be noted that ‘full size’ refers to the modification of all pixels in an image. The PDH curves of the cover image (the plaintext image) and the steganography image (the image with embedded data) are analyzed to detect any visible histogram deviations, which would suggest the influence of data embedding on the image.

As illustrated in Figure 8, the results of the embedding test with 10 KB show that there is essentially no discernible difference between the cover image and the steganography image curves. This similarity suggests that the pixel difference histogram is not substantially altered by embedding a small amount of data 10 KB, thereby enabling the image to maintain its original structural and visual characteristics. This result shows that small data sizes are well suited for secure and imperceptible steganographic applications, as they preserve the integrity of the pixel relationships within the cover image.

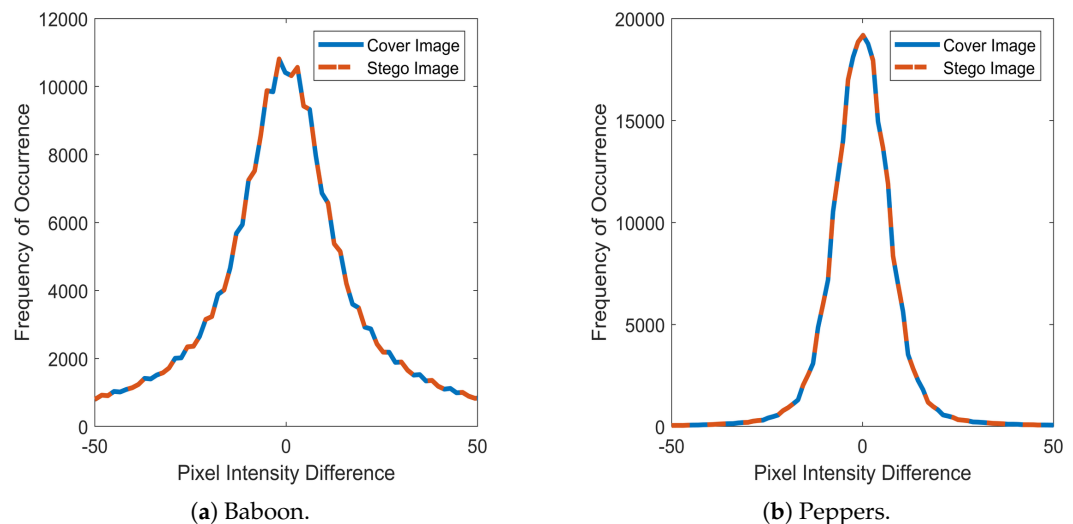


Figure 8. PDH security analysis at 10 KB embedding capacity: (a) Baboon image, (b) Peppers image.

As can be observed in Figure 9, the curves of the cover and steganography images exhibit a slight divergence in the half-full-size embedding (embedding into half of the pixels). However, this discrepancy suggests a minor influence on the pixel relationships within the image; the variation is negligible and does not significantly disrupt the histogram pattern. This subtle deviation implies that the PDH curve maintains a high degree of similarity to the original even with a moderate embedding size, indicating the method’s robustness against moderate data payloads.

The differences between the cover and steganography curves become more apparent when data is embedded into the full image size, as illustrated in Figure 10. This increase in deviation indicates that the pixel difference histogram is more significantly influenced by higher embedding capacities, as expected. However, the steganography curve still bears a striking resemblance to the original cover curve, despite the greater variation, to the extent that the discrepancies could be regarded as minor. This implies that the visual quality of the image remains acceptable in most applications despite the fact that full-size embedding introduces detectable modifications. The impact is relatively contained, confirming that the proposed method can resist attacks.

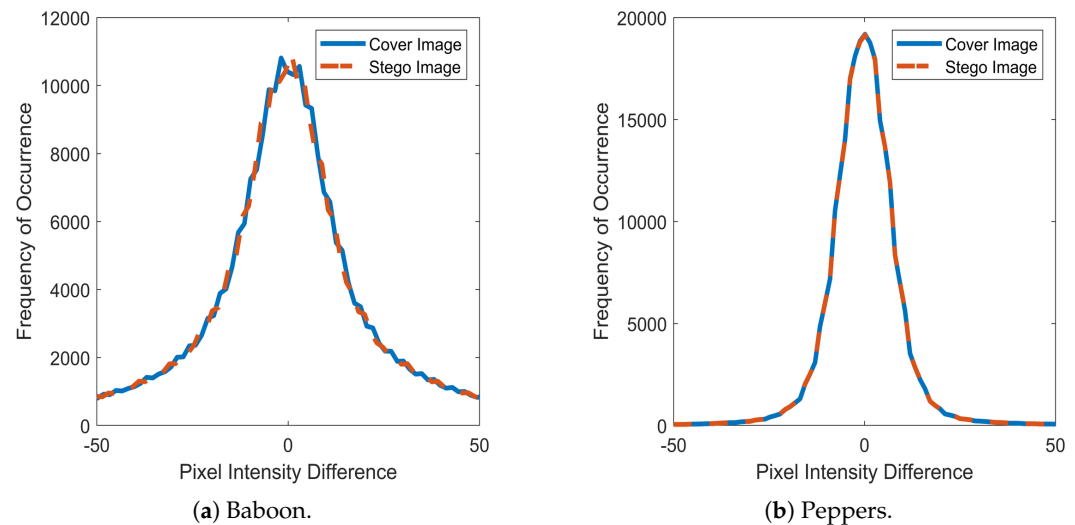


Figure 9. PDH security analysis at half-full-size embedding capacity: (a) Baboon image, (b) Peppers image.

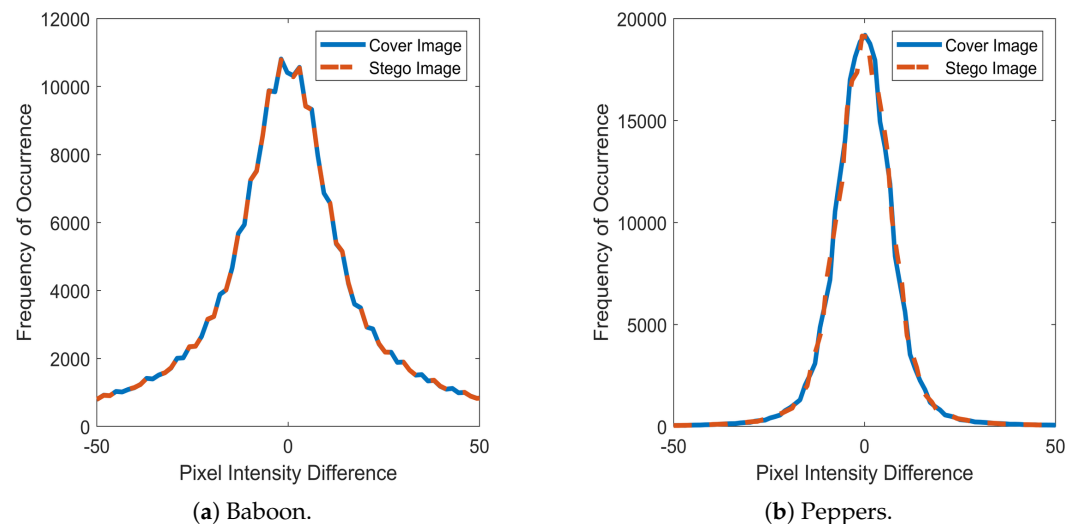


Figure 10. PDH security analysis at full-size embedding capacity: (a) Baboon image, (b) Peppers image.

3.5. Effect of Changing the B-LSB Values on the PSNR

Altering the value of b-LSB by embedding multiple bits per pixel within the LSB method of image steganography also impacts the PSNR value. The parameter b defines the number of bits modified in each pixel of the cover image. When $b = 1$, only the least significant bit is adjusted, resulting in minimal perceptual distortion. Conversely, as b increases, the changes become more pronounced, potentially negatively affecting image quality. Figure 11 illustrates the degradation in PSNR as the number of modified bits increases, specifically for the Baboon image.

As shown in Figure 11, the PSNR decreases linearly as b increases. Meanwhile, the MSE also rises, indicating a greater discrepancy between the original and steganographic images. This highlights a trade-off between higher embedding capacity and visual imperceptibility.

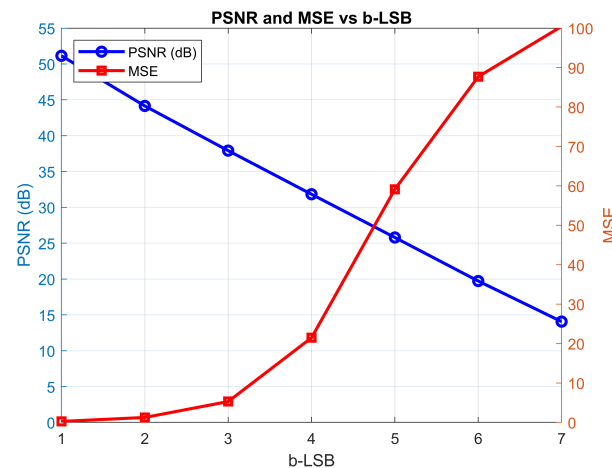


Figure 11. Degradation of PSNR and MSE as b -LSB increases for the Baboon image.

Notably, when b exceeds 3, the MSE increases sharply, indicating that higher bit plane embedding introduces substantial distortion. We stress that this behavior is well established in the steganography literature and is not claimed as a novel finding in this work. Here, we report it only as an empirical confirmation under the proposed hybrid pipeline (SPN + chaotic permutation + LSB embedding) and to justify the choice of a practical operating range (capped at $b \leq 4$) that balances data hiding capacity and steganographic image quality in our evaluation.

4. Conclusions and Future Work

This paper presents a novel approach to secure data transmission by integrating encryption and steganography. The proposed method encrypts the message using the CLM and SPN encryption algorithms, employing row and column permutations to improve security. The encrypted data is then embedded into a cover image using LSB substitution steganography, ensuring imperceptibility while maintaining robust protection. The experimental results demonstrate that the proposed technique effectively balances image quality, embedding capacity, and security. The PSNR values indicate minimal perceptual distortion, while the efficient execution time further underscores the method's practicality. Pixel difference analysis also supports the method's effectiveness, as histogram variations remain nearly imperceptible when embedding data up to half the size of the cover image. We also note that a data compression approach can also be used to handle a large payload by reducing its size and reading the data as a binary file in uint8 format, thereby increasing the effective embedding capacity.

In future research, adaptive embedding techniques can be leveraged by integrating deep learning models or transformers to identify optimal embedding regions. This dynamic approach can further enhance imperceptibility and security. Additionally, exploring multidimensional chaotic systems could further improve unpredictability and robustness. Another possible direction for further research is to adapt the proposed method towards creating a highly efficient hybrid video encryption and steganography algorithm which utilizes the same lightweight chaotic map for random pixel selection.

The use of advanced steganalysis tools is also necessary to verify the imperceptibility of the approach and its level of resistance to steganalysis attacks. Some such tools include XuNet (structural design of conventional neural networks) [33,34], while more are being developed. There is also a need for a unified benchmark that supports fair and transparent evaluation of different steganography methods.

Author Contributions: Conceptualization, A.A. and A.M.; methodology, A.A., M.A.H., and A.M.; software, A.A.; validation, A.A. and M.A.H.; formal analysis, A.A., M.A.H., and A.M.; investigation, A.M.; resources, A.M.; data curation, A.A.; writing—original draft preparation, A.A., M.A.H., and A.M.; writing—review and editing, A.A., M.A.H., and A.M.; visualization, A.A., M.A.H., and A.M.; supervision, A.M.; project administration, A.M.; funding acquisition, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study is available from the corresponding author upon reasonable request. The data is not publicly available due to proprietary reasons.

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

References

1. Abdullah, A.M. Advanced encryption standard (AES) algorithm to encrypt and decrypt data. *Cryptogr. Netw. Secur.* **2017**, *16*, 11.
2. Riaz, M.; Dilpazir, H.; Naseer, S.; Mahmood, H.; Anwar, A.; Khan, J.; Benitez, I.B.; Ahmad, T. Secure and fast image encryption algorithm based on modified logistic map. *Information* **2024**, *15*, 172. [\[CrossRef\]](#)
3. Kaur, M.; Kumar, V. A comprehensive review on image encryption techniques. *Arch. Comput. Methods Eng.* **2020**, *27*, 15–43. [\[CrossRef\]](#)
4. Gong, L.; Qiu, K.; Deng, C.; Zhou, N. An image compression and encryption algorithm based on chaotic system and compressive sensing. *Opt. Laser Technol.* **2019**, *115*, 257–267. [\[CrossRef\]](#)
5. Song, Y.; Zhu, Z.; Zhang, W.; Yu, H.; Zhao, Y. Efficient and secure image encryption algorithm using a novel key-substitution architecture. *IEEE Access* **2019**, *7*, 84386–84400. [\[CrossRef\]](#)
6. Giriya, R.; Singh, H. A new Substitution-Permutation Network Cipher Using Walsh Hadamard Transform. In *Proceedings of the 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), Gurgaon, India, 12–14 October 2017*; IEEE: New York, NY, USA, 2017; pp. 168–172.
7. Arif, J.; Khan, M.A.; Ghaleb, B.; Ahmad, J.; Munir, A.; Rashid, U.; Al-Dubai, A.Y. A novel chaotic permutation-substitution image encryption scheme based on logistic map and random substitution. *IEEE Access* **2022**, *10*, 12966–12982. [\[CrossRef\]](#)
8. Shannon, C.E. Communication theory of secrecy systems. *Bell Syst. Tech. J.* **1949**, *28*, 656–715. [\[CrossRef\]](#)
9. Biryukov, A. Substitution–Permutation (SP) Network. In *Encyclopedia of Cryptography and Security*; van Tilborg, H.C.A., Jajodia, S., Eds.; Springer: Boston, MA, USA, 2011; p. 1268.
10. Belazi, A.; Abd El-Latif, A.A.; Belghith, S. A novel image encryption scheme based on substitution-permutation network and chaos. *Signal Process.* **2016**, *128*, 155–170. [\[CrossRef\]](#)
11. Nashat, D.; Mamdouh, L. An efficient steganographic technique for hiding data. *J. Egypt. Math. Soc.* **2019**, *27*, 57. [\[CrossRef\]](#)
12. Karakus, S.; Avci, E. A new image steganography method with optimum pixel similarity for data hiding in medical images. *Med Hypotheses* **2020**, *139*, 109691. [\[CrossRef\]](#)
13. ALabaichi, A.; Al-Dabbas, M.A.A.K.; Salih, A. Image steganography using least significant bit and secret map techniques. *Int. J. Electr. Comput. Eng.* **2020**, *10*, 935–946. [\[CrossRef\]](#)
14. Butt, K.K.; Li, G.; Khan, S.; Manzoor, S. Fast and efficient image encryption algorithm based on modular addition and SPD. *Entropy* **2020**, *22*, 112. [\[CrossRef\]](#) [\[PubMed\]](#)
15. Huang, F.; Zhong, Y.; Huang, J. Improved Algorithm of Edge Adaptive Image Steganography Based on LSB Matching Revisited Algorithm. In *Digital-Forensics and Watermarking: 12th International Workshop, IWDW 2013, Auckland, New Zealand, 1–4 October 2013*; Revised Selected Papers; Springer: Berlin, Germany, 2014; pp. 19–31.
16. Emam, M.M.; Aly, A.A.; Omara, F.A. An improved image steganography method based on LSB technique with random pixel selection. *Int. J. Adv. Comput. Sci. Appl.* **2016**, *7*, 361–366. [\[CrossRef\]](#)
17. Sheidaee, A.; Farzinvas, L. A Novel Image Steganography Method Based on DCT and LSB. In *Proceedings of the 2017 9th International conference on Information and Knowledge Technology (IKT), Tehran, Iran, 18–19 October 2017*; IEEE: New York, NY, USA, 2017; pp. 116–123.
18. Gnanitha, G.; Swetha, A.; Sai Teja, G.; Sri Vasavi, D.; Lakshmi Sirisha, B. Review on Image Steganography Transform Domain Techniques. In *Proceedings of the Conference of Innovative Product Design and Intelligent Manufacturing System*; Springer: Singapore, 2022; pp. 501–512.
19. Shah, K.; Kaul, S.; Dhande, M.S. Image steganography using DWT and data encryption standard (DES). *Int. J. Sci. Res. (IJSR)* **2014**, *3*, 372–376.

20. Banik, B.G.; Bandyopadhyay, P.; Samir, K. A DWT method for Image Steganography. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2013**, *3*, 1–7.
21. Saraswathy, P.G.; Siddaiah, R. Randomness properties of sequence generated using logistic map with novel permutation and substitution techniques. *Int. J. Electr. Comput. Eng.* **2023**, *13*, 4369–4378. [[CrossRef](#)]
22. Mishra, R.; Dutta, S.; Okade, M.; Mahapatra, K. Substitution Permutation Network Based Lightweight Ciphers with Improved Substitution Layers for Secure IoT Applications. In *Proceedings of the 2021 2nd International Conference on Range Technology (ICORT)*, Balasore, India, 5–6 August 2021; IEEE: New York, NY, USA, 2021; pp. 1–6.
23. Alaklabi, A.; Munir, A.; Hafeez, M.A.; Khattak, M.A.K. Z-crypt: Chirp Z-transform-based image encryption leveraging chaotic logistic maps and substitution permutation network. *IEEE Access* **2024**, *12*, 123401–123422. [[CrossRef](#)]
24. Chandramouli, R.; Memon, N. Analysis of LSB Based Image Steganography Techniques. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, Thessaloniki, Greece, 7–10 October 2001; IEEE: New York, NY, USA, 2001; Volume 3, pp. 1019–1022.
25. University of Southern California. *USC-SIPI Image Database. Volume 3: Miscellaneous*. Available online: <https://sipi.usc.edu/database/database.php?volume=misc> (accessed on 15 May 2025).
26. Hameed, M.A.; Hassaballah, M.; Aly, S.; Awad, A.I. An adaptive image steganography method based on histogram of oriented gradient and PVD-LSB techniques. *IEEE Access* **2019**, *7*, 185189–185204. [[CrossRef](#)]
27. Alanzy, M.; Alomrani, R.; Alqarni, B.; Almutairi, S. Image steganography using LSB and hybrid encryption algorithms. *Appl. Sci.* **2023**, *13*, 11771. [[CrossRef](#)]
28. Hore, A.; Ziou, D. Image Quality Metrics: PSNR vs. SSIM. In *Proceedings of the 2010 20th International Conference on Pattern Recognition, Istanbul, Turkey, 23–26 August 2010*; IEEE: New York, NY, USA, 2010; pp. 2366–2369.
29. Wu, D.C.; Tsai, W.H. A steganographic method for images by pixel-value differencing. *Pattern Recognit. Lett.* **2003**, *24*, 1613–1626. [[CrossRef](#)]
30. Riley, L.; Mandal, J.; Das, D. Colour image steganography based on pixel value differencing in spatial domain. *Int. J. Inf. Sci. Tech. (IJIST) Vol* **2012**, *2*, 83–93.
31. Shen, S.Y.; Huang, L.H. A data hiding scheme using pixel value differencing and improving exploiting modification directions. *Comput. Secur.* **2015**, *48*, 131–141. [[CrossRef](#)]
32. Zhang, X.; Wang, S. Vulnerability of pixel-value differencing steganography to histogram analysis and modification for enhanced security. *Pattern Recognit. Lett.* **2004**, *25*, 331–339. [[CrossRef](#)]
33. Xu, G.; Wu, H.Z.; Shi, Y.Q. Structural design of convolutional neural networks for steganalysis. *IEEE Signal Process. Lett.* **2016**, *23*, 708–712. [[CrossRef](#)]
34. Yang, R.; Yang, Y.; Zhou, L.; Meng, X. A Steganalysis Method Based on Relationship Mining. *Electronics* **2025**, *14*, 4347. [[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.