symmetry

MDPI

Article

Novel, Fast, Strong, and Parallel: A Colored Image Cipher Based on SBTM CPRNG

Ahmad Al-Daraiseh [1], Yousef Sanjalawe [2], Salam Fraihat [3,*] and Salam Al-E'mari [4]

1 Computer Science Department, Faculty of Information Technology, American University of Madaba, Amman 11821, Jordan; a.daraiseh@aum.edu.jo
2 Cybersecurity Department, Faculty of Information Technology, American University of Madaba, Amman 11821, Jordan; y.sanjalawe@aum.edu.jo
3 Artificial Intelligence Research Center (AIRC), College of Engineering and Information Technology, Ajman University, Ajman P.O. Box 346, United Arab Emirates
4 Information Security Department, Faculty of Information Technology, University of Petra, Amman 11196, Jordan; salam.ammari@uop.edu.jo
* Correspondence: s.fraihat@ajman.ac.ae

Abstract: Smartphones, digital cameras, and other imaging devices generate vast amounts of high-resolution colored images daily, stored on devices equipped with multi-core central processing units or on the cloud. Safeguarding these images from potential attackers has become a pressing concern. This paper introduces a set of six innovative image ciphers designed to be stronger, faster, and more efficient. Three of these algorithms incorporate the State-Based Tent Map (SBTM) Chaotic Pseudo Random Number Generator (CPRNG), while the remaining three employ a proposed modified variant, SBTMPi. The Grayscale Image Cipher (GIC), Colored Image Cipher Single-Thread RGB (CIC1), and Colored Image Cipher Three-Thread RGB (CIC3) showcase the application of the proposed algorithms. By incorporating novel techniques in the confusion and diffusion phases, these ciphers demonstrate remarkable performance, particularly with large colored images. The study underscores the potential of SBTM-based image ciphers, contributing to the advancement of secure image encryption techniques with robust random number generation capabilities.

Keywords: chaos theory; chaotic systems; color image encryption; colored image; cryptography; image encryption; security

Citation: Al-Daraiseh, A.; Sanjalawe, Y.; Fraihat, S.; Al-E'mari, S. Novel, Fast, Strong, and Parallel: A Colored Image Cipher Based on SBTM CPRNG. *Symmetry* **2024**, *16*, 593. https://doi.org/10.3390/sym16050593

Academic Editors: Karl Hess and Jie Yang

Received: 18 March 2024
Revised: 22 April 2024
Accepted: 30 April 2024
Published: 10 May 2024

1. Introduction

The rapid progress and widespread adoption of contemporary digital communication and network technologies have revealed significant potential in improving online data storage and electronic data exchange. However, ensuring the protection of confidential information is equally essential, leading to the continuous prioritization of network security and data integrity [1]. Consequently, scientists have implemented appropriate safety measures to enhance visibility and prevent security vulnerabilities. Since a large portion of multimedia content shared and stored online consists of images, it becomes necessary to guarantee the confidentiality and authenticity of digital images through the utilization of image encryption techniques. The process of image encryption involves utilizing a mathematical algorithm to convert the original image into an intricate format, making it challenging to interpret. This transformation strengthens the image's resistance to security attacks [2–4]. Image encryption finds practical applications in diverse fields such as medical imaging, telemedicine, business, biometric authentication, and military communication. To address the security demands of these applications, various image encryption techniques have been developed, including digital watermarking techniques [5], image scrambling methods [6], image steganography [7], and image cryptography [8].

In recent decades, there has been a significant rise in interest surrounding the application of chaos in cryptography. This interest stems from the fundamental property of chaos,

which makes it sensitive to initial conditions and produces deterministic data sets that resemble randomness. By leveraging chaos-based cryptographic models, researchers have developed pioneering techniques for designing efficient image encryption systems. These systems showcase remarkable attributes in multiple areas, such as speed, cost-effectiveness, computational power, computational overhead, complexity, vulnerability, and more. The utilization of chaotic systems in cryptography offers notable benefits, such as sensitivity to initial values, strong randomness, and resistance to cracking [9]. Consequently, chaotic systems have gained significant attention and been extensively studied for image encryption purposes [10–12].

In 1989, mathematician Matthews pioneered the integration of chaos into encryption systems and introduced the concept of chaotic cryptography [13]. Compared to traditional image encryption methods, encryption techniques based on chaotic systems demonstrate higher efficiency. Consequently, researchers have proposed a multitude of image encryption schemes that leverage chaotic systems [5,14,15]. In a color image encryption (CIE) algorithm, it is common practice to encrypt the three components of R, G, and B separately and then merge them to generate a color image [16]. Alternatively, one can create a unified grayscale image by dividing and encrypting the three components, allowing for the restoration of the original size [17]. After extensive research conducted by countless scholars, the field of color image CIE has made significant progress and achieved a more refined state. The amalgamation of chaotic systems and scrambling methods within the scrambling–diffusion framework has proven effective in meeting the majority of encryption needs.

Nevertheless, there is still scope for enhancing the existing encryption algorithms. However, when it comes to proposing algorithms for image encryption, scholars have primarily utilized chaos theory, optical transformation, DNA encoding, and compressed sensing. These theories serve as the foundation for developing a wide range of image encryption algorithms that modify the positions and values of pixels. Despite the abundance of proposed image encryption algorithms, most of them concentrate on single grayscale images, raising doubts about their security and efficiency. To address these concerns, this paper introduces a new algorithm that strives to improve both the security and efficiency of image encryption. In order to enhance both security and efficiency, a novel CIE algorithm is introduced. The core contributions of this paper are outlined as follows:

- Enhanced parallel execution: The proposed algorithms were developed to enable efficient parallel execution across all phases.
- Innovations in confusion and diffusion techniques: The proposed algorithms incorporate original approaches within the confusion and diffusion processes, resulting in streamlined and parallel execution.
- Robust random number generation: These algorithms make use of SBTM, which not only is highly efficient but also boasts remarkable strength in random number generation.
- Streamlined encryption methods for grayscale and color images: The encryption and decryption processes for images have been optimized to achieve swift performance, with encryption times as low as 1 ms for small images and 1 s for larger images (100 MB).
- Heightened sensitivity to key and image modifications: Through a single round of encryption, the cipher images exhibit extraordinary responsiveness to alterations in both the encryption key and the original image.

The structure of this paper is organized as follows: Section 2 provides a comprehensive analysis of previous studies. Section 3 introduces a new CIE algorithm. Section 4 showcases the experimental results and provides detailed analyses. Finally, Section 5 concludes the paper by summarizing the findings.

2. Related Works

In this section, a synthetic analysis of the latest colored image encryption techniques is presented. Numerous algorithms for encrypting colored images have been proposed,

showcasing variations in terms of their effectiveness and robustness. The encryption of images is the procedure of securing them from unauthorized access utilizing a secret key. Digital visual data are structured in the form of rectangular frames consisting of individual elements referred to as pixels, each assigned a numerical value. With the progress of information technologies, digital images, encompassing diverse formats like medical images, grayscale images, color images, binary images, and others, have become increasingly prevalent in applications, storage, and transmission. Consequently, protecting this type of information has emerged as a crucial challenge [18]. Considering the distinctive attributes of image data, numerous encryption algorithms have been proposed, employing various technologies, including [19–27].

A color image encryption technique was proposed by Aqeel ur Rehman et al., utilizing an exclusive-OR operation with DNA complementary rules. The basis of this approach was the integration of chaos theory and the SHA-256 hash function to modify the initial conditions and control parameters of the chaotic system [28]. The color image's three channels were transformed into a one-dimensional vector and sorted according to the chaotic sequence generated via the Piecewise Linear Chaotic Map. This permuted array was then divided into three parts, representing each color channel, and independently permuted using Lorenz's chaotic system. Following the dual permutation, each pixel in every channel was encoded independently using deoxyribonucleic acid (DNA) bases in a chaotic manner. Notably, the algorithm's novelty lies in substituting each pixel in a channel with an exclusive-OR operation based on DNA complementary rules. Multiple DNA rules were employed, repeating this operation for a random number of cycles in a cyclic fashion. The selection of DNA rules at the beginning of this cyclic operation and its continuation depended on Chen's chaotic sequence. Extensive simulated experiments demonstrated the algorithm's exceptional encryption performance achieved within a single round.

In their work, Seyedzadeh et al. [29] proposed an innovative image encryption algorithm that utilized chaos theory, employing a Coupled Two-Dimensional Piecewise Nonlinear Chaotic Map (CTP-NCM) and a masking process specifically designed for encrypting color images. Through computer simulations, it was demonstrated that the algorithm achieved a high level of security while maintaining exceptional speed in practical color image encryption applications. The algorithm incorporated a 256-bit external secret key to generate the initial conditions and parameters of the CTP-NCM. Experimental results substantiated the superior security performance of the proposed algorithm in comparison to other existing algorithms. Tong et al. presented a rapid algorithm for encrypting color images, utilizing a four-dimensional chaotic system as its foundation [30]. To enhance the encryption algorithm's complexity and key space, the researchers introduced an innovative method for designing four-dimensional chaotic systems based on classical equations from three-dimensional chaotic systems. The authors proposed a new pseudo-random sequence generator. The generator utilizes bits from the four values produced via the four-dimensional chaotic systems to generate three keys instead of one, leveraging the generated sequence to enhance the speed of the image encryption process. The authors employed row-major and column-major techniques to diffuse the original image, and they employed a cat map with a parameter to scramble the image pixels and achieve the desired encryption effect. Extensive simulations and security analyses confirmed that the proposed encryption algorithm demonstrated remarkable performance in terms of security, robustness, and high encryption speed.

Li et al. introduced an image encryption algorithm that leveraged hyper-chaos, utilizing a 5D, multi-wing, hyper-chaotic system in which the key stream was derived from the original image [31]. The algorithm integrated pixel-level and bit-level permutations to enhance the cryptosystem's security. Furthermore, a diffusion operation was employed to modify the pixels. Theoretical analysis and numerical simulations substantiated the algorithm's ability to provide high security and reliability for image encryption. In a separate study, researchers proposed a simple and effective chaotic system by combining two existing one-dimensional chaotic maps (seed maps) [18]. Simulations and performance eval-

uations demonstrated that this system generated one-dimensional chaotic maps with larger chaotic ranges and improved chaotic behaviors compared to the seed maps. The chaotic system was then utilized in a novel image encryption algorithm to address multimedia security. Notably, with the utilization of the same set of security keys, the algorithm generated distinct encrypted images for the same original image in each encryption process. Experimental investigations and security analyses verified the algorithm's exceptional performance in image encryption and its resilience against various attacks. In addition, El-Latif et al. presented an innovative scheme for encrypting color images based on a quantum chaotic system [32]. The method involved applying new substitution total automorphism in the integer wavelet transform, specifically scrambling only the Y (luminance) component of the L frequency sub-band. Subsequently, two diffusion modules were created by combining the features of horizontally and vertically adjacent pixels using a quantum chaotic map. In the final stage, a substitution/confusion process was generated by employing an intermediate chaotic key stream image derived from the quantum chaotic system. Thorough security and performance analyses were conducted through extensive experimental investigations, showcasing the exceptional characteristics of the proposed color image encryption method, including robust security and satisfactory performance. Comparative evaluations favored the proposed scheme in most cases.

Furthermore, Liu et al. proposed an image encryption scheme based on chaos using bijection [33]. In this algorithm, the entire color image was diffused using XOR operations for a random number of rounds. Each color component was separated into blocks of equal size. A bijective function, denoted as f: B→S, was established between the block set B and the S-box set S. The corresponding 8 × 8 S-box was dynamically generated via the Chen system, incorporating variable conditions. The encrypted image was obtained by substituting each block with the corresponding paired S-box. Numerical simulations and security analyses indicated that the scheme exhibited the potential for practical application in image encryption. Additionally, Mazloom et al. introduced a novel image encryption algorithm based on chaos, specifically utilizing a coupled nonlinear chaotic map (CNCM), to encrypt color images [34]. The algorithm employed symmetric key cryptography with a stream cipher structure. To enhance the security of the algorithm, a 240-bit secret key was used to generate the initial conditions and parameters of the chaotic map through algebraic transformations applied to the key. These transformations, combined with the nonlinear and coupling characteristics of the CNCM, resulted in an improved level of security within the cryptosystem. In this study, the image size and color components were incorporated to achieve higher security and complexity, thereby significantly increasing the approach's resistance to known and chosen–plaintext attacks. The algorithm's efficacy and security were validated through various experimental and statistical analyses, as well as key sensitivity tests, demonstrating its suitability for real-time image encryption and transmission.

Kadir et al. presented a color image encryption scheme that employed a couple of hyperchaotic Lorenz systems [35]. The scheme introduced a novel approach by injecting impulse signals randomly into the coupled Lorenz system during iterations to enhance the complexity of the trajectory. Six sequences of state variables were generated to encrypt the R, G, and B components using bitwise operations such as XOR and a left or right cyclic shift. The utilization of six initial values and multiple indeterminate impulse signals expanded the cryptosystem's key space, providing resistance to exhaustive attacks, including those from quantum computers. Simulations demonstrated the stability of the mean encryption speed, which was influenced by the hardware equipment and the algorithm employed. Statistical analysis confirmed the high effectiveness of the proposed image encryption algorithm. Existing color image encryption techniques using chaotic techniques have several main limitations.

- Security: Some existing algorithms are vulnerable to attacks such as brute-force, statistical, and differential attacks. These vulnerabilities can compromise the security of encrypted images.

- Key space: Chaotic systems often have a limited key space, making them susceptible to exhaustive key search attacks. If the key space is small, it is easier for an attacker to try all possible keys and decrypt the encrypted image.
- Robustness: Some algorithms are not robust enough to withstand common signal processing operations or geometric transformations applied to the encrypted image. This can lead to a loss of information or a decrease in the quality of the decrypted image.
- Speed: Chaos-based encryption algorithms can sometimes be computationally intensive, leading to slower encryption and decryption speeds. This can limit their practical use in real-time applications or scenarios requiring quick processing.
- Resistance to attacks: Existing algorithms may not provide sufficient resistance to advanced attacks such as chosen-plaintext attacks or known-plaintext attacks. These attacks exploit the algorithm's vulnerabilities and can potentially reveal the original content of the encrypted image.

Addressing these limitations is crucial to ensuring the development of more secure and efficient color image encryption algorithms based on chaotic techniques. In addition to the limitations mentioned above, there are also other challenges that need to be addressed in order to develop more secure and efficient color image encryption algorithms using chaotic techniques. These challenges include the following:

- Developing more secure chaotic systems with a larger key space.
- Developing more robust chaotic-based encryption algorithms that are resistant to common signal processing operations and geometric transformations.
- Developing more efficient chaotic-based encryption algorithms that can achieve high encryption and decryption speeds.
- Developing chaotic-based encryption algorithms that are resistant to advanced attacks, such as chosen-plaintext attacks and known-plaintext attacks.

The development of more secure and efficient color image encryption algorithms using chaotic techniques is an active area of research. By addressing the challenges mentioned above, it is possible to develop more secure and efficient color image encryption algorithms that can be used to protect sensitive data.

3. Proposed Technique

This section provides an overview of the proposed image ciphers. A total of six versions have been developed, with three of them employing the SBTM [36] random number generator, while the other three adopt a modified version utilizing digits of PI as its state. This modification demonstrates the flexibility of SBTM, showcasing its ability to yield strong random numbers with diverse initial values. The following presents a brief description of the first three algorithms that were developed:

1. A Grayscale Image Cipher (GIC), a cipher that can only be used with grayscale images.
2. Colored Image Cipher Single Thread RGB (CIC1), a cipher that can be used with color images and uses a single thread to handle the red, green, and blue channels of a sub-image.
3. Colored Image Cipher Three Thread RGB (CIC3), a cipher that can be used with color images and that uses three threads to handle the red, green, and blue channels of a sub-image.

The preceding three algorithms rely on the original SBTM random number generator, while the remaining three ciphers share identical structures but implement a modified version of SBTM. All six versions adhere to a common logic; therefore, we discuss GIC in detail, and subsequently, we explore the distinctions present in the other variants. The description of a step-by-step implementation can be found in the appendices.

3.1. Image Encryption Methods and Ciphers

In this article, we focus on a number of strategies to produce image ciphers that are very strong and efficient. In particular, we consider the following:

1. The use of strong and efficient CPRNG: Random number generators are essential components in ciphers. They determine the cipher strength and time of execution. We decided to use SBTM CPRNG for its strong sequences and speed. Details about SBTM and the proposed variant, SBTMPi, are provided in the next section.
2. Parallel processing: Currently, almost all CPUs are multi-core, even those of smartphones and tablets. Encrypting an image in a sequential fashion takes a long time. Due to the nature of random number generators and the sequences they produce, many of the ciphers operate sequentially. In this article, however, we create four arrays of random numbers in advance and use them to shuffle and encrypt an image. This method allows for parallel processing utilizing all cores of the CPU, which results in a huge speed gain.
3. Number of random numbers required: Many of the image ciphers produce as many random numbers as the number of pixels in an image. Depending on the efficiency of the PRNG used, this process requires a large amount of time, especially for large images. In this article, we only produce a small number of random numbers to encrypt the image; for example, for images of size (1000 * 1000), we produce only 4000 random numbers. The numbers are then utilized in a smart way to provide strong encryption, as evidenced by the results shown in the Implementation and Evaluation section.
4. Efficient shuffling of pixels: Another aspect of image encryption that usually consumes time is the shuffling process. Shifting and rolling columns and rows of an image is time-consuming and gets worse with large images. We utilized a clever method to calculate new indices of a plain image pixel and store the encrypted pixel in the calculated indices in the encrypted image. This method avoided the actual shifting of pixels and allowed for parallel processing as well.
5. Sensitivity to change: We developed an efficient method that provides very strong sensitivity to changes in either plain images or their keys. A change of one to any of the plain image's pixels leads to a drastic change in the encrypted image. The method used produces a digest of the image that can be used for an integrity check as a byproduct.

As mentioned above, a total of six ciphers are proposed, namely GIC, CIC1, and CIC3, which utilize SBTM CPRNG, and another three that uilize a proposed variant of SBTM, which are discussed in the next section. The overall processes of the six ciphers are summarized below, while a description of their detailed implementation is delegated to the appendices. The main steps followed in all six ciphers are listed below:

1. Load the image.
2. Divide the image into a number of sub-images.
3. Generate strong keys: as can be seen in Figure 1, $uk1$ and US are used to initialize SBTM. Ninety-seven values are generated, and then $uk2$ is used to replace the control variable of SBTM, and so on until all keys are used. The state of SBTM is now influenced by all user input. $ek1$ is assigned to the next random value. Ninety-seven values are generated, and the last one is assigned to $ek2$. This step is repeated for $ek3$ to $ek6$.
4. Use a number of threads to initialize the required arrays: GIC has only two arrays to hold random double values, $Diff1$ size r and $Diff2$ size c. Hence, only two threads are needed.
5. Use a number of threads to calculate a digest called ES: based on the number of sub-images, create as many threads to calculate ES. The process is basically the sum of all the elements of the image after adding a random value to it and multiplying it with the sum of two random values from $Diff1$ and $Diff2$ as shown in the equation below. This process produces a simple digest that is affected by any modification to any pixel of the original image. $\sum_{i=0,j=0}^{r,c} (I[i,j] + Diff2[j]) * (Diff1[i] + Diff2[j])$.
6. Use a number of threads to initialize another set of arrays utilizing the calculated digest: GIC has only two arrays to hold random integer values, $Roll1$, size r, and

*Roll*2, size c. Hence, only two threads are needed. Note that these two arrays hold randomly ordered indices.

7. Use a number of threads to encrypt the image: based on the number of sub-images, create as many threads to encrypt the image. Each thread is assigned one sub-image. The encryption step is rather simple. It XORs pixels from the plain image with four values from the four arrays, $Diff1$, $Diff2$, $Roll1$, and $Roll2$. Shift emulation is done here, through which new indices are calculated for any given pixel in the encrypted image, and the plain image pixel at the calculated indices is encrypted.
8. Save the digest and the encrypted image to file.

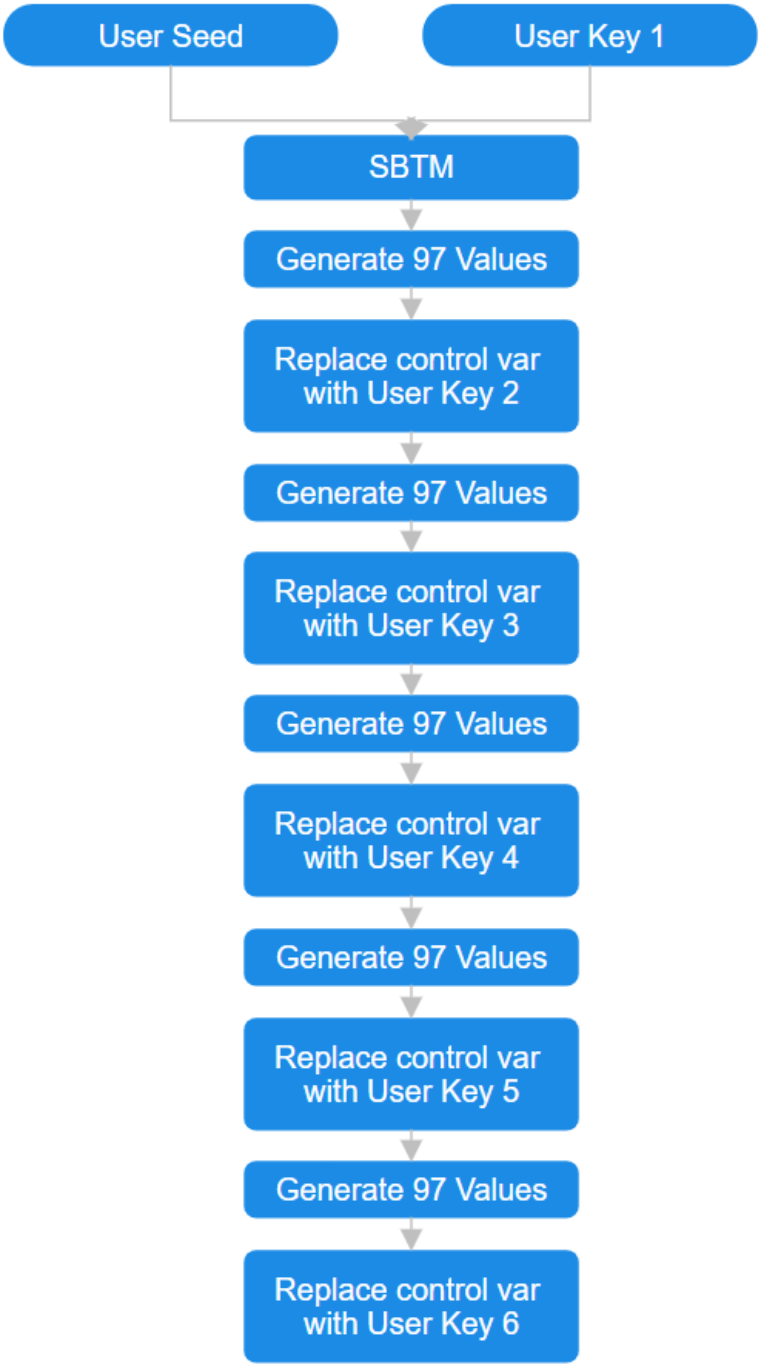


Figure 1. Key generation process.

Figure 2 summarizes the encryption process. A step-by-step example can be found in Appendix D.4.

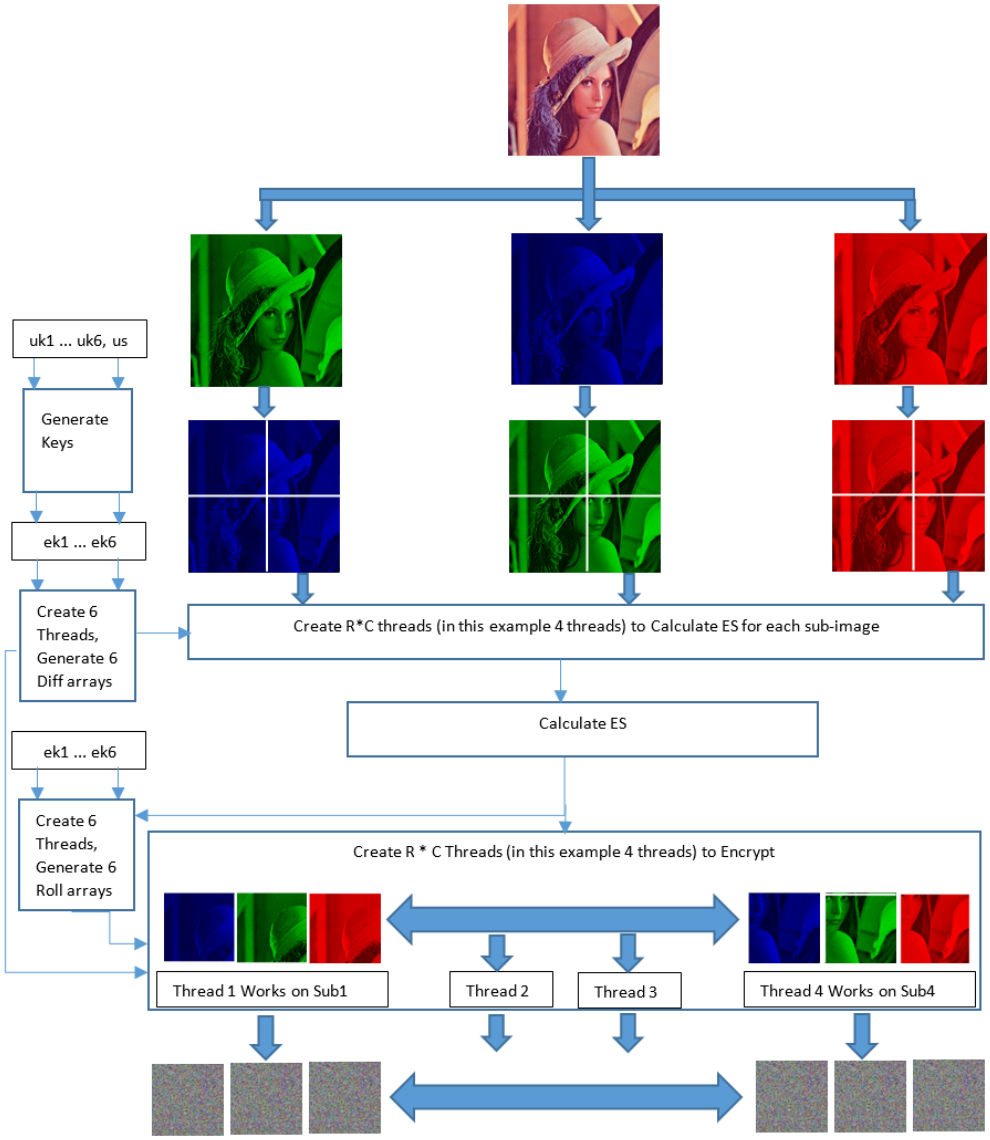


Figure 2. CIC1 encryption process.

3.2. SBTM and SBTMPi

SBTM is the CPRNG proposed in [36]. It was specifically designed to generate sequences with exceptional statistical properties and a high degree of security while maintaining a low computational cost. It utilizes a modified 1D chaotic tent map with enhanced attributes to produce chaotic sequences. SBTM utilizes the well-known tent map function and adds to it a circular array to serve as a state. The initial values of the state are square roots of prime numbers. The main functionality of SBTM is shown in Equation (1). With every random number generated, an element of the state is updated. An automatic reset of state values is applied to keep them within a specific range. SBTM was thoroughly tested and proved to produce extremely strong sequences with a low computational time. The size of the state array was chosen to be five. In the original article, many sizes were tested, and they all produced strong sequences. Five, however, kept the size of the generator small. The values produced from Equation (1) can be greater than one; only the fraction is returned to keep the random values between zero and one. In [36], it was stated that other well-chosen values can be used for the initial state without impacting the strength of the generator. In this

article, we propose SBTMPi, which uses digits from the fraction part of PI. The digits were multiplied by 100 and then by the constant (**e**). The state array used is of length 13, as shown below: State of $SBTMPi$ = {271.8281828459045, 1087.312731383618, 271.8281828459045, 1359.1409142295227, 2446.4536456131405, 543.656365691809, 1630.969097075427, 1359.1409142295227, 815.4845485377135, 1359.1409142295227, 2174.625462767236, 2446.4536456131405, 1902.7972799213317 }. GICP$_i$, CIC1P$_i$, and CIC3P$_i$ use the proposed CPRNG, namely SBTMPi.

$$r_{n+1} = \begin{cases} r_n \times \mu \times \text{state}[i], & \text{if } n < 0.5 \\ (1 - r_n) \times \mu \times \text{state}[i], & \text{otherwise} \end{cases} \quad (1)$$

3.3. Image Decryption Process

The decryption process is simpler and faster than the encryption process due to the fact that the digest does not have to be calculated, as it is impeded in the encrypted file. The general procedure followed for all six ciphers can be summarized in the following steps:

1. Load *ES*, then load the encrypted image.
2. Divide the image into a number of sub-images.
3. Generate strong keys: a similar process as that of the encryption.
4. Use a number of threads to initialize the required arrays
5. Use a number of threads to decrypt the image, which is the opposite of the encryption step.
6. Save the decrypted image to file.

The detailed algorithms of the decryption processes are presented in the appendices.

3.4. Analytical Discussions

In this section, the reason for developing new image ciphers and the design decisions made are discussed.

Why develop a new image cipher? There are at least three reasons that can be used. (1) With the increasing number of cyber attacks and privacy issues, stronger ciphers need to be developed. (2) The current hardware used in PCs, laptops, and smartphones uses multi-core chips that need to be utilized. (3) The increasing number of images taken, and the enormous size of such images, makes a clear case for faster and more efficient ciphers.

Why use SBTM? A random number generator is an essential part of most if not all image ciphers. Many of the used PRNGs in the literature suffer from weaknesses such as a limited range of the control variable, non-uniform distribution, being slow, and/or weak sequences of random numbers. On the contrary, SBTM is a new CPRNG that is very strong and efficient, as demonstrated in [36].

Why use PI as the initial seed? Practically any value could be used. To avoid simple and suspicious values, common constants are utilized. PI and the square roots of prime numbers are irrational and very random in nature.

Why use six keys, not seven, ten, or three, for example? SBTM uses only one control variable as a key by default. The users may choose weak values such as 1 or 5 as a key. To enlarge the key space of the cipher and to make it harder for an attacker, more keys are needed. Of course, there needs to be a balance between usability and security. While using fewer keys makes it more usable it also makes it weaker. On the other hand, using more keys makes it less usable but stronger. Six keys seem to achieve such balance. This number of keys has been used in the literature as well [37].

Why generate digest *ES*? There are two reasons here. (1) The user may choose a simple seed, such as 0.1 or 0.2, which makes it easy to guess. (2) Since, usually, images are encrypted pixel by pixel after being confused, a change in the original image will only impact one pixel in the encrypted image. For these two reasons, the proposed algorithms generate a strong digest dependent on the following: the user seed US, the generated Diff arrays, and the pixel value. The Diff arrays make the position of the pixel somewhat unique;

i.e., if a white image were used, and only pixel values were used, changing pixel [0, 0] by adding one to it would produce the same digest as changing pixel [0, 1] or [5, 5], and so on. Utilizing a pair of random doubles will produce a different seed if the change is made to a pixel at a different location, given that all pixels have the same value. Experimental results show this impact.

Why generate encryption keys other than the user keys $uk1$ through 6? The user may choose simple keys such as all zero or one, etc. Despite the fact that SBTM is very strong, it will generate the same sequence if given the same key and seed. This will definitely negatively impact the strength of the encrypted image. To avoid such a situation, the algorithm uses the user keys to generate strong and different keys $EK1$ through 6 that are used to generate the random sequences.

Why do we divide the image? It is very clear that time is very important, and the current hardware is capable. Using a single thread to encrypt or decrypt an image is a waste of time and resources. The proposed algorithms give the user a choice to decide on the number of sub-images to be used and, hence, reduce time and improve hardware utilization. Experimental results show a huge gain when parallelizing the process.

Why emulate the shuffling of the image? Shuffling the image before encryption is essential in almost all image ciphers. This process takes a large amount of time, especially for large images of 10 MP and above. The goal is to make this process more efficient. A novel approach to calculate where the location of the pixel will be after shuffling, and then encrypting that pixel is proposed. It saves a large amount of time when dealing with large images, as shown in the results.

4. Implementation and Verification

All six algorithms, including SBTM and SBTMPi, were implemented in Java on a laptop equipped with an Intel Core i7 twelfth-generation processor and 16 GB of RAM, running Windows 11. Java was chosen for its concurrent programming capabilities and user-friendly nature. Numerous images were subjected to testing, but only the results for those listed in Table 1 are reported due to space constraints. Throughout all experiments, the following values were consistently used: $US = 0.3$, and $uk1$ through $uk6$ were 3.4, 5.6, 2.1, 7.8, 6.1, and 9.4, respectively. The time results presented are averages from 100 runs. The results were compared with those from the prior literature [37–50].

Table 1. Used images and their sources. Sources were accessed on 2 May 2024.

	Name	Size	Source
1	Colored Lenna.png	512 * 512	https://en.wikipedia.org/wiki/Lenna
2	Colored Peppers.tiff	512 * 512	https://sipi.usc.edu/database/
3	Colored Baboon.tiff	512 * 512	https://sipi.usc.edu/database/
4	bernard-hermant-nHRXNv2qeDE-unsplash.jpg	6000 * 4000	https://unsplash.com
5	White.png	512 * 512	Created
6	Black.png	10,000 * 10,000	Created
7	Gray jocelyn-morales-ybDvbCvh9Ro-unsplash.jpg	2918 * 2832	https://unsplash.com
8	Gray danny-M7l0CS4yBsY-unsplash.jpg	4032 * 3024	https://unsplash.com
9	Gray Lenna.png	512 * 512	Created from original

In this section, the evaluation of the proposed algorithms concerning key space, statistical analysis, sensitivity, and time complexity is explored. For statistical analysis, the histogram, correlation, and entropy are assessed. To examine sensitivity, the performances of NPCR, UACI, and BACI are calculated. Following that, the obtained results are compared with results from the prior literature [37–50].

4.1. Key Space Analysis

To effectively thwart brute force attacks, it is crucial to have a sufficiently large key space. In cryptography, a computational complexity exceeding 128 bits is considered adequately secure. In this section, we analyze the key space of the proposed algorithms to

ensure that it significantly surpasses the 128-bit threshold. The secret parameters used for this analysis are the user seed (US), and the six user keys, $(UK1 \ldots UK6)$. Each parameter is represented as a double with 15 digits of precision, equivalent to 52 bits. Equation (2) demonstrates that the keyspace indeed spans 364 bits, far exceeding the required minimum of 128 bits. As a result, the key meets this crucial security requirement.

$$\text{key length} = 7 * 52 = 364 \text{ bits}, \quad \text{Key space} = 2^{364} \tag{2}$$

4.2. Statistical Analysis

In the literature, three commonly employed statistical analysis tools are typically taken into account: histograms, correlation, and entropy. Histogram analysis involves evaluating the distribution of pixel values in images. To assess the relationship between the original and encrypted images, the correlation coefficients of the horizontal, vertical, and diagonal directional features are calculated. Additionally, Shannon's entropy is utilized to demonstrate the randomness of the encrypted images.

4.2.1. Histogram Analysis

The significance of the histogram lies in its ability to reveal crucial insights into the randomness of an encrypted image. For an encrypted image to be secure, the distribution of pixel values should be uniform, preventing adversaries from extracting any valuable information through statistical attacks. Samples of histograms of the original images, as well as those of the encrypted images, are shown in Figure 3. It is very clear that the histogram of the encrypted image is uniformly distributed, even for images with very little entropy, such as white.png and black.png, as can be seen in the following figures.

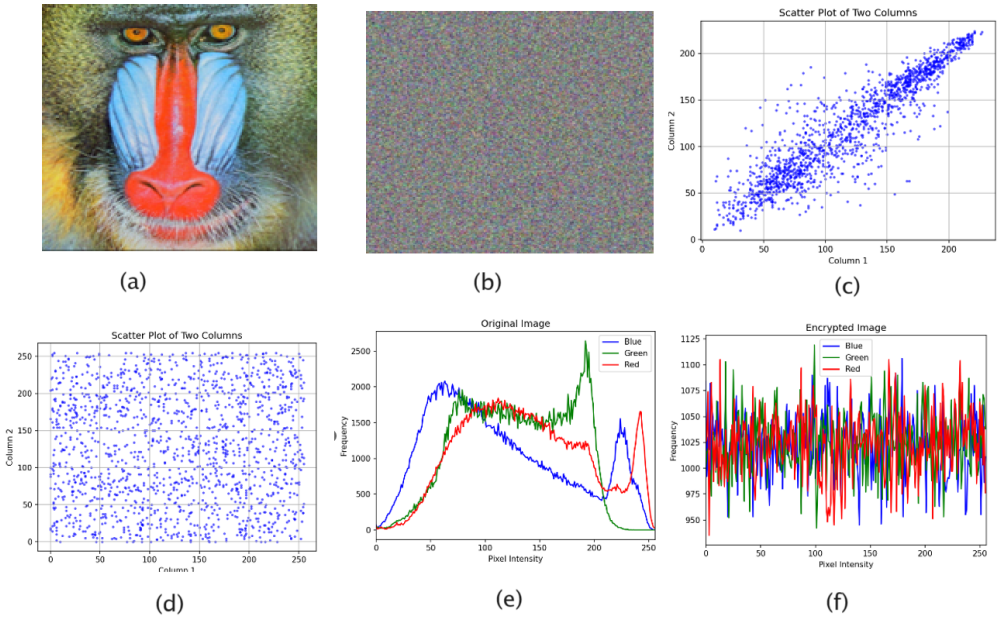


Figure 3. baboon.tiff (512, 512) colored: (**a**) original image, (**b**) encrypted image, (**c**) correlation of two adjacent columns in (**b**), (**d**) correlation of two adjacent columns in (**a**), (**e**) histogram of (**a**), and (**f**) histogram of (**b**).

4.2.2. Correlation Analysis

In typical images, there is often a strong correlation between adjacent pixels. Nevertheless, an effective encryption scheme should generate cipher images with a mini-

mal correlation between neighboring pixels to conceal the underlying image information. The correlation coefficient can be computed using Equation (3).

$$r_{xy} = \frac{E(x - E(x))E(y - E(y))}{\sqrt{D(x)D(y)}} \tag{3}$$

where x and y are two adjacent pixels selected for calculation. $E(x)$ and $E(y)$ are the expectation of x and y, while $D(x)$ and $D(y)$ are the variances of x and y. For both the plain and cipher images, pairs of all pixels were considered in three directions: horizontal, vertical, and diagonal. The correlation coefficient was then computed, and the outcomes are displayed in Table 2.

The analysis reveals that the plain images exhibit a significant correlation between adjacent pixels, whereas the cipher images generated via GIC, CIC1, and CIC3 demonstrate notably low correlation values. Figures 4–7 visually illustrate the correlation between two adjacent columns in both the plain and cipher images. The figure vividly demonstrates that pixels in the cipher images appear to be randomly distributed, while those in the same columns of the plain image exhibit a pronounced correlation.

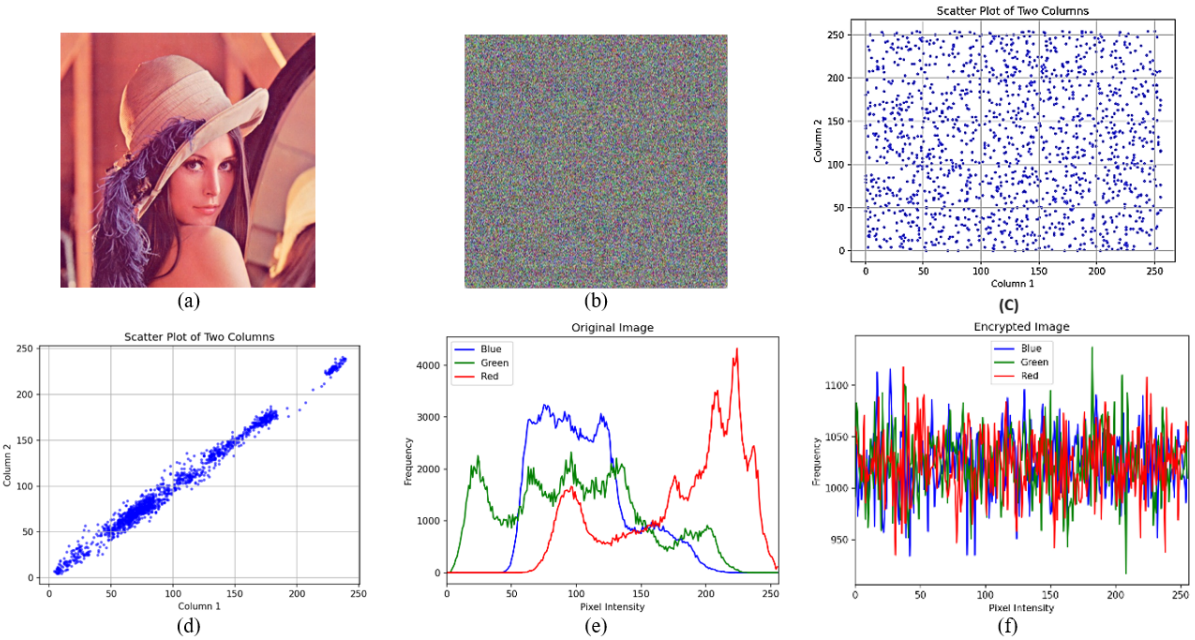


Figure 4. Lenna.png colored: (**a**) original image, (**b**) encrypted image, (**c**) correlation of two adjacent columns in (**b**), (**d**) correlation of two adjacent columns in (**a**), (**e**) histogram of (**a**), and (**f**) histogram of (**b**).

4.2.3. Information Entropy

Information entropy is a mathematical property used to measure the randomness of an information source. It is calculated based on Equation (4).

$$H(s) = -\sum_{i=0}^{2^N-1} P(s_i)\log_2(P(s_i)) \tag{4}$$

where s is the information source; N is the number of bits used to represent s; and $P(si)$ is the probability of the occurrence of symbol x_i. The entropy value of a genuinely random source with 256 gray levels is expected to be 8. After calculating the information entropy for all images, the results were compiled in Table 3. The entropy values for all cipher images surpassed 7.999, while for Black.jpg, they exceeded 7.99999. This indicates that GIC, CIC1, and CIC3 effectively generate cipher images that withstand entropy analysis, ensuring robust security.

Table 2. Correlation values produced by GIC, CIC1, and CIC3.

	Lenna.png	**Baboon.tiff**	**Bernard-hermant-nHRXNv2qe DEunsplash.jpg https**	**White.png**	**Black.png**	**Peppers.tiff**	**Jocelyn–Morales**	**Danny**	**Lennag.png**
Correlation of Original									
Horizontal Correlation Coefficient	0.971928377	0.866543103	0.968008701	1	1	0.976770914	0.982348334	0.999713197	0.976212614
Vertical Correlation Coefficient	0.985029604	0.758730154	0.959743177	1	1	0.97920518	0.987638886	0.99973431	0.986572767
Diagonal Correlation Coefficient	0.959331116	0.726188678	0.935853441	1	1	0.963934813	0.974356812	0.999537899	0.964138041
Correlation of Encrypted—SBTM									
Horizontal Correlation Coefficient	2.439×10^{-3}	4.979×10^{-3}	-2.219×10^{-4}	5.78×10^{-4}	-1.48×10^{-5}	3.734×10^{-3}	2.234×10^{-4}	6.27×10^{-5}	-1.17×10^{-3}
Vertical Correlation Coefficient	-6.538×10^{-4}	-4.314×10^{-3}	-2.83×10^{-5}	1.483×10^{-4}	-5.86×10^{-5}	-2.07×10^{-3}	3.103×10^{-4}	1.855×10^{-4}	-1.805×10^{-3}
Diagonal Correlation Coefficient	-4.148×10^{-3}	-2.290×10^{-3}	4.36×10^{-5}	7.277×10^{-4}	-2.995×10^{-4}	-1.56×10^{-3}	3.118×10^{-4}	5.35×10^{-6}	-1.15×10^{-3}
Correlation of Encrypted—SBTMPi									
Horizontal Correlation Coefficient	-4.553×10^{-4}	0.976770914	-1.238×10^{-4}	-4.374×10^{-4}	-1.719×10^{-3}	-2.653×10^{-3}	-3.986×10^{-4}	-2.153×10^{-4}	3.239×10^{-3}
Vertical Correlation Coefficient	-1.143×10^{-3}	0.97920518	-3.24×10^{-5}	-9.716×10^{-4}	7.546×10^{-4}	2.852×10^{-3}	-4.177×10^{-4}	-3.295×10^{-4}	2.597×10^{-3}
Diagonal Correlation Coefficient	-1.291×10^{-3}	0.963934813	-1.506×10^{-4}	6.831×10^{-4}	-1.050×10^{-3}	5.539×10^{-4}	-2.19×10^{-5}	-1.609×10^{-4}	-3.304×10^{-4}

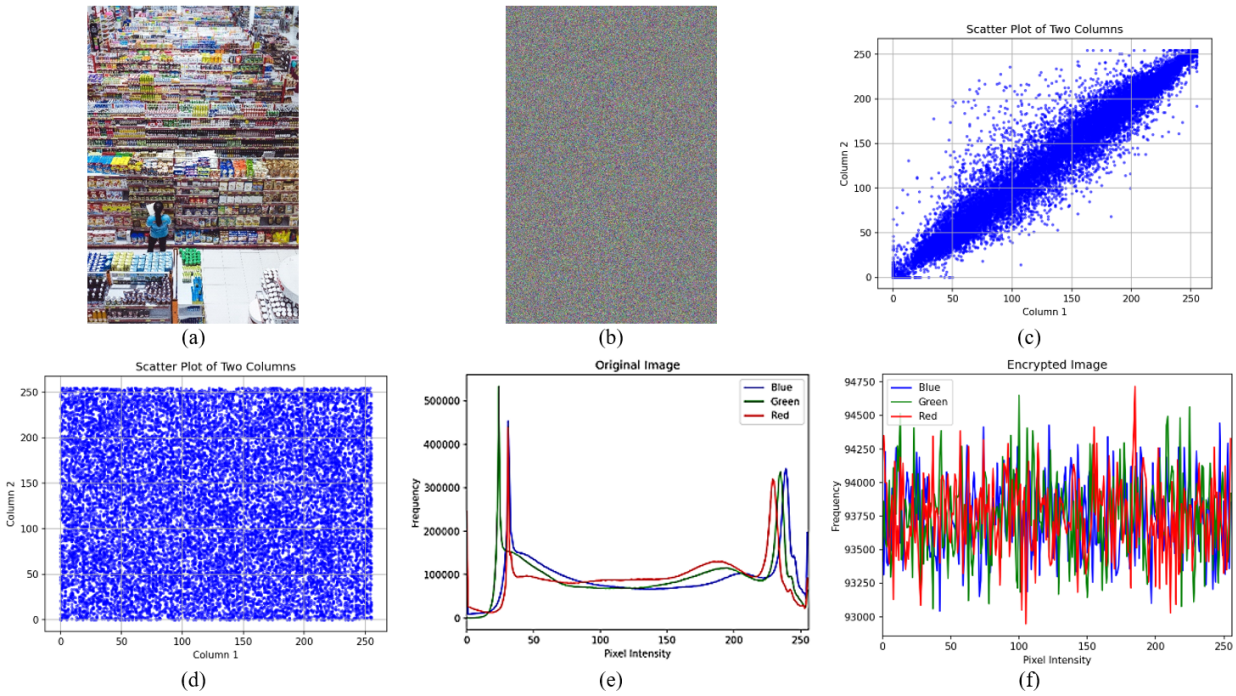


Figure 5. bernard-hermant-nHRXNv2qeDE-unsplash.jpg (6000 * 4000): (**a**) original image, (**b**) encrypted image, (**c**) correlation of two adjacent columns in (**b**), (**d**) correlation of two adjacent columns in (**a**), (**e**) histogram of (**a**), and (**f**) histogram of (**b**).

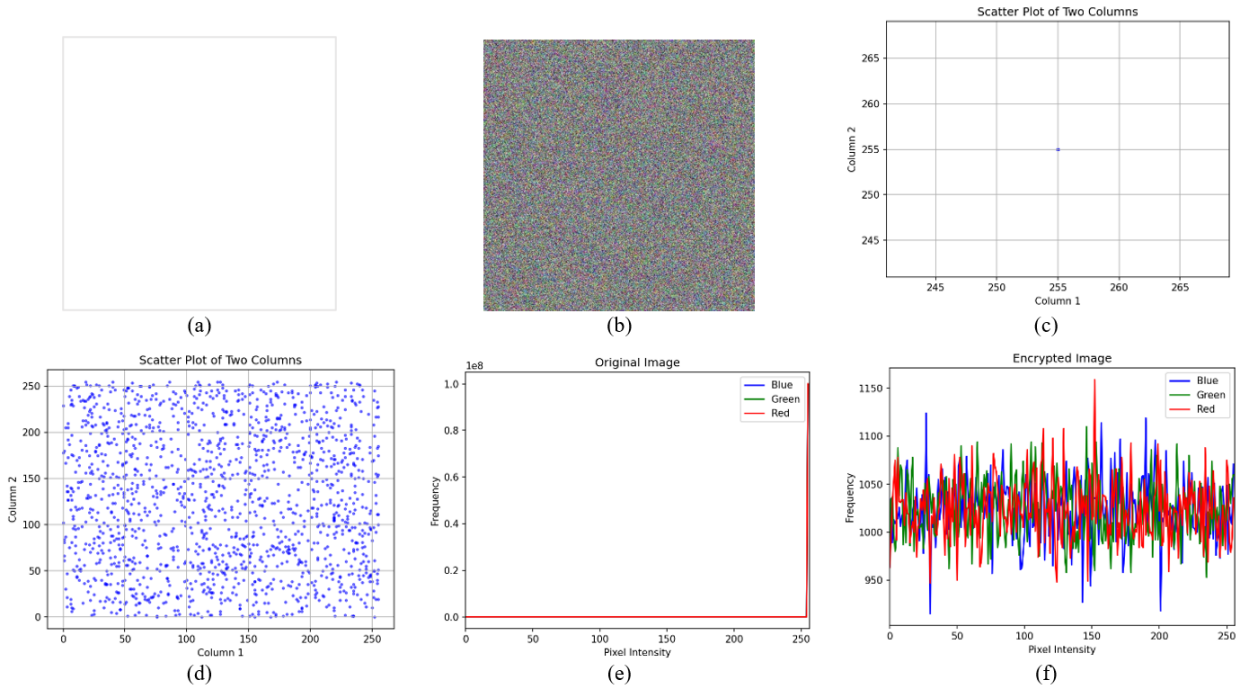


Figure 6. white.png: (**a**). original image, (**b**) encrypted image, (**c**) correlation of two adjacent columns in (**b**), (**d**) correlation of two adjacent columns in (**a**), (**e**) histogram of (**a**), and (**f**) histogram of (**b**). (512 * 512).

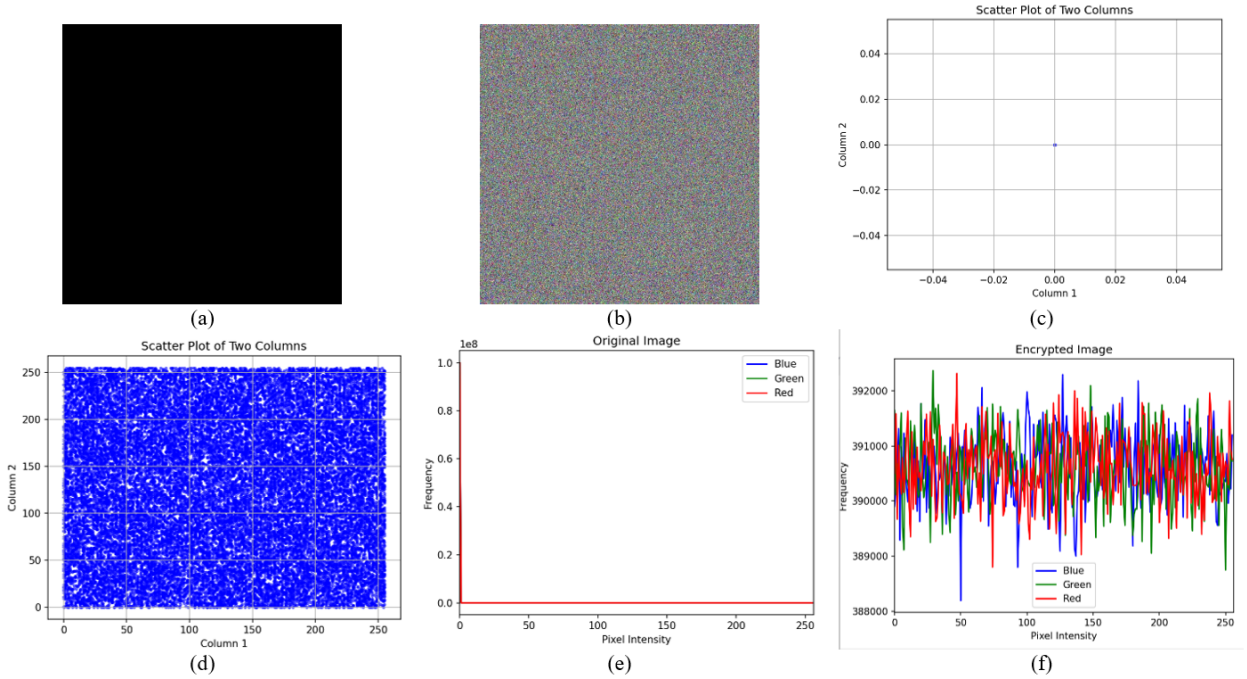


Figure 7. BLACK.png (10,000 * 10,000): (**a**) original image, (**b**) encrypted image, (**c**) correlation of two adjacent columns in (**b**), (**d**) correlation of two adjacent columns in (**a**), (**e**) histogram of (**a**), and (**f**) histogram of (**b**).

4.2.4. Sensitivity Analysis

Claude Shannon identified two crucial properties of a secure cipher: confusion and diffusion. Confusion involves the relationship between the cipher text and the encryption key, while diffusion pertains to the relationship between the cipher text and the plain text. In other words, any change to a character of the key or the plain text should result in multiple characters of the cipher text being altered. Contemporary cryptographic algorithms often employ a substitution–permutation (SP) network as a straightforward approach to achieving both confusion and diffusion. The S-Box used in AES is a popular method for achieving substitution. However, this type of substitution can require large memory usage or lead to high computational complexity.

To address this limitation, the proposed algorithms adopt a novel approach to computing the digest *ES* and another novel approach to emulating pixel shifting. Furthermore, they utilize all the cores available in the CPU in the main steps to enhance efficiency. To demonstrate that these algorithms yield cipher images satisfying Shannon's properties, three evaluation metrics were calculated for all test images: NPCR (Number of Pixel Change Rate), UACI (Unified Average Changing Intensity), and BACI (Blocked Average Changing Intensity). NPCR and UACI were computed for all three color channels between the following pairs of images:

1. The original image and the cipher image.
2. Two cipher images: the cipher image and one generated after adding one to pixel [0, 0, 0] of the original image.
3. Two cipher images: the cipher image and one generated after adding 10^{-15} to the user seed *US* or any of the user keys.

BACI values were calculated for two pairs of images, cases 2 and 3 of the above. NPCR, UACI, and BACI were calculated using Equations (5), (7), and (9).

Table 3. Entropy of original images vs encrypted images.

	Lenna.png	Baboon.tiff	Bernard-hermant-nHRXNv2qeD Eunsplash.jpg https	White.png	Black.png	Peppers.tiff	Jocelyn–Morales	Danny	Lennag.png
Entropy of Original									
Red Channel	6.968426946	7.752217172	7.784196631	0	0	7.058305579	1.072629341	7.762109322	7.457123508
Green Channel	7.594037916	7.474431586	7.765855155	0	0	7.496253345			
Blue Channel	7.253102357	7.706671843	7.81265644	0	0	7.338826961			
Entropy of Encrypted—SBTM									
Red Channel	7.999292363	7.999361961	7.999992751	7.999290305	7.999998056	7.999265701	7.999981573	7.999985503	7.999312018
Green Channel	7.999262644	7.999348141	7.999992205	7.999319074	7.999998152	7.999383295			
Blue Channel	7.999293901	7.999293719	7.999992849	7.999278883	7.999998185	7.999368216			
Entropy of Encrypted—SBTMPi									
Red Channel	7.999288148	7.058305579	7.999992368	7.999311326	7.999292731	7.99926848	7.999979201	7.99998552	7.999227501
Green Channel	7.999253219	7.496253345	7.999992296	7.999322162	7.999341788	7.999290724			
Blue Channel	7.999311461	7.338826961	7.999992204	7.999410048	7.999238474	7.99924015			

$$NPCR = \frac{\sum_{j=1}^{H}\sum_{i=1}^{W} D(i,j)}{WH} \times 100\% \tag{5}$$

$$D(i,j) = \begin{cases} 1, & \text{if } c_1(i,j) \neq c_2(i,j) \\ 0, & \text{if } c_1(i,j) = c_2(i,j) \end{cases} \tag{6}$$

$$UACI = \sum_{j=1}^{H}\sum_{i=1}^{W} \frac{|c_1(i,j) - c_2(i,j)|}{P \times WH} \tag{7}$$

$$UACI(P_1, P_2) = \frac{1}{mN}\sum_{i=1}^{M}\sum_{j=1}^{N} \frac{|P_1(i,j) - P_2(i,j)|}{255} \times 100\%. \tag{8}$$

$$BACI(P_1, P_2) = \frac{1}{(M-1)(N-1)} \sum_{k=1}^{(M-1)(N-1)\frac{m_k}{255}} \tag{9}$$

The comprehensive details and expected values for the equations mentioned above, specifically when comparing the plain image with the cipher image and when comparing cipher images with each other, can be found in [51], where the expected values are shown in Table 4. The relevant results are presented in Table A1.

Table 4. Theoretical values of NPCR, UACI, and BACI indexes

Images	Image Size	NPCR (%)	UACI (%)	BACI (%)
Between two random images	256 * 256	99.6094	33.435	26.7712
Between Lena and random image	256 * 256	99.6094	28.5923	21.3268
Between Mandrill and random image	256 * 256	99.6094	27.4210	20.1118
Between Peppers and random image	256 * 256	99.6094	29.5685	22.1874
Between All-black and random image	256 * 256	99.6094	50.0000	33.4635
Between All-white and random image	256 * 256	99.6094	50.0000	33.4635

Table A1 demonstrates that the calculated values for the three indexes closely align with the theoretical values provided in Table 4. For NPCR, all calculated values were consistently greater than 99.6 across all three color channels. Regarding UACI between cipher images, the values were always above 33.4 for all channels, except for peppers.tiff, where it was 33.39456. However, when considering the average UACI value across all three channels, it exceeded 33.4. Furthermore, UACI between the original images and the cipher images was in close agreement with the results in Table A1, particularly for the Lenna, Peppers, White, and Black images. The BACI results were even more promising than those of UACI. According to [51], the obtained BACI values were around the theoretical value of 26.7712 for all cipher images. These findings indicate that the proposed algorithms effectively generate cipher images that exhibit a high level of randomness and sensitivity to changes in the original image or any of the keys used.

4.2.5. X^2 Test

The X^2 test is a quantitative method used to assess the extent of deviation in the pixel distribution of an image from a perfectly uniform distribution. The calculation is represented in Equation (10), where pi denotes the frequency of pixel i in the image, and p is the average frequency of all pixels, computed for $(m \times n)/256$, where m and n represent the height and width of the image, respectively.

$$C^2 = \sum_{i=0}^{255} \frac{(P_i - \bar{P})^2}{\bar{P}} \tag{10}$$

A smaller X^2 value indicates a closer adherence of the image to a uniform distribution, signifying a more uniform pixel distribution. Table 5 displays the X^2 values for both the plain and cipher images. Notably, the X^2 values of the cipher images are significantly lower compared to those of the plain images. The X^2 values for the proposed algorithms are generally around 250, which is the expected value. Particularly, for six of the nine images, it was less than 250.

Table 5. X^2 values.

	SBTM		**SBTMi**	
Image	X^2 Original	X^2 Encrypted	X^2 Original	X^2 Encrypted
Lenna.png	236,580.67	259.6582	236,580.67	[259.0983]
Baboon.tiff	100,859.73	240.7334	100,859.73	[248.74544]
Bernard-hermant-nHR XNv2qeDE-unsplash.jpg https	7,984,348	244.85942	7,984,348	[256.04306]
White.png	261,120	254.85449	261,120	[236.34863]
Black.png	2.55×10^{10}	258.76398	2.55×10^{10}	[256.9603]
Peppers.tiff	339,975.53	210.97806	339,975.53	[266.7067]
Jocelyn-morales	1.7×10^{9}	239.7806	1.7×10^{9}	[237.87653]
Danny	4,219,045.5	243.0403	219,045.5	[244.72905]
Lennag.png	159,456.94	245.01074	159,456.94	[280.40234]

4.2.6. Digest ES Sensitivity

Each of the six algorithms produces a digest ES, which is crucial for both the encryption and the decryption processes. ES serves as a highly random and sensitive parameter used in conjunction with the user seed, US, to seed both SBTM and SBTMPi in order to generate the random numbers used during encryption and decryption. Table A1 presents the distinct values obtained when encrypting the original image, an image with only one pixel modified, and when the encryption key is altered for all test images. Notably, the values are remarkably different even when the changes introduced to the image or the key are minimal. This highlights the robustness and sensitivity of ES, making it a vital component in ensuring the security and effectiveness of the encryption process. It is worth noting that ES is written to file. This is very safe, as no parameters can be extracted from it, and before using it, it is mixed with the user seed.

4.2.7. SBTM vs. SBTMPi-Based Ciphers

This article introduces SBTMPi as a proposed extension of SBTM, showcasing the remarkable robustness and flexibility of the original CPRNG. By carefully selecting different values for the state of SBTM, the strength of the generated sequences remains unaffected. The encryption of all test images was conducted using GICPi and CIC3Pi. The evaluation results, displayed in Tables A2 and A3, illustrate that the generated cipher images are equally strong, if not better, compared to those produced using GIC and CIC3.

For instance, in many cases, the X^2 value of the encrypted black and white images was lower, signifying improved performance. Additionally, the entropy of many encrypted images exceeded 7.99999, further validating their robustness. Overall, all other evaluation metrics exhibited equally favorable outcomes, highlighting the efficacy of SBTMPi and its ability to produce secure and strong cipher images.

4.2.8. CIC1 vs. CIC3

CIC1 and CIC3 are two algorithms employed to encrypt colored images. The key difference between them lies in their approach to processing sub-images during encryption. CIC3 utilizes three threads to encrypt each sub-image, while CIC1 employs a single thread.

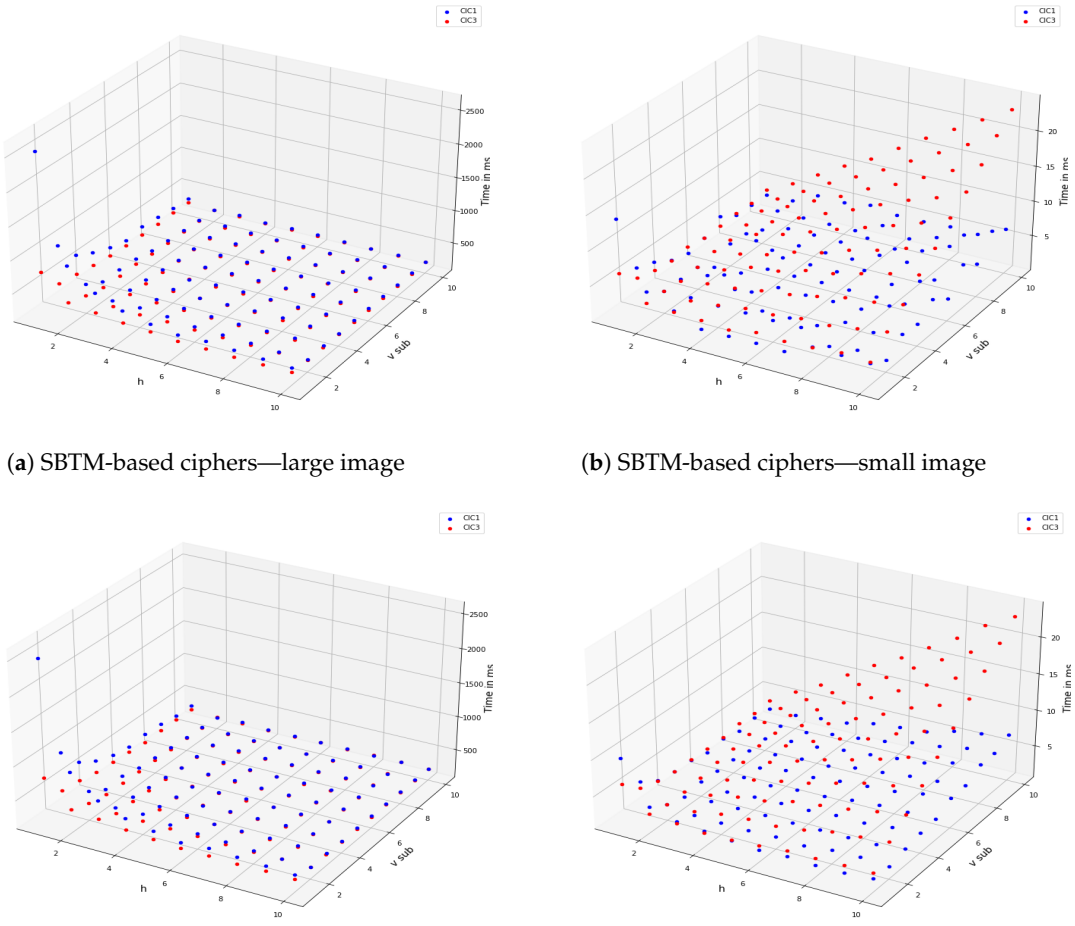
To determine which algorithm is more suitable and to illustrate the significance of dividing the image into various sub-images determined according to the values of R and C, a comprehensive experiment was conducted. Two images, one small (peppers.tiff) and

one large (bernard-hermant-nHRXNv2qeDE-unsplash.jpg), were selected for testing. Each image was encrypted using both CIC1 and CIC3 with a varying number of sub-images, where R and C ranged from 1 to 10. The entire process was repeated 100 times, and the average encryption time was recorded.

A similar experiment was conducted to compare CIC1Pi and CIC3Pi. Figure 8 and Tables A2 and A3 illustrate the results obtained from these experiments. Notably, when the number of generated sub-images was one, CIC3 performed nearly three times faster than CIC1 for both images. However, as the number of sub-images increased, the behavior differed. For the large image, the encryption time was quite similar, but CIC3 demonstrated better performance with a minimum average of 246.7 ms compared to 256.1 ms for CIC1. Conversely, for the small image, CIC1 performed better as the number of sub-images grew. The minimum average was 1.5 ms (when $R = 4$ and $C = 1$) for CIC1 and 2.5 ms (when $R = 5$ and $C = 1$) for CIC3. It is worth noting that both algorithms required more time as the number of sub-images increased.

Similar outcomes were observed when comparing CIC1Pi and CIC3Pi. In conclusion, for large images, CIC3 with a large number of sub-images is recommended, while for small images, CIC1 with few sub-images is more suitable for efficient encryption.

Figure 8 shows the encryption time vs. the number of sub-images in the following: A. SBTM-based CIC1 is compared with CIC3 for a large image. B. SBTM-based CIC1 is compared with CIC3 for a small image. C. SBTMPi-based CIC1 is compared with CIC3 for a large image. D. SBTMPi-based CIC1 is compared with CIC3 for a small image.



(**a**) SBTM-based ciphers—large image

(**b**) SBTM-based ciphers—small image

(**c**) SBTMPi-based ciphers—large image

(**d**) SBTMPi-based ciphers—small image

Figure 8. CIC1 and CIC3 performance against the number of sub-images used. h and v represent the horizontal and vertical numbers of sub-images.

4.2.9. Comparisons

In this section, we compare the results obtained from various metrics with those reported by other researchers. It is crucial to acknowledge that this comparison may not be entirely accurate in many cases due to differences in the images used and the evaluation metrics employed. Nonetheless, presenting this comparison is essential to showcase how the proposed algorithms fare against those introduced by other researchers. Tables 6–9 show comparison results between the proposed encryption techniques with state-of-the-art methods when encrypting an image, which offers valuable insights into the quality and effectiveness of the image encryption process using the proposed technique. As depicted in Table 6, the proposed technique demonstrates superior performance compared to the approach presented in [43] when encrypting the "lenna.png" colored image. This superiority is evident in several aspects, including a substantial reduction in encryption time, from 1670 ms to 2.38 ms consumed for the proposed encryption technique compared to the time consumed for [43] to encrypt the "lenna.png" colored image. Additionally, there was a notable improvement in the correlation between the original and encrypted images, as well as in the UACI and NPCR metrics for channel "0".

Moreover, Table 7 presents a more extensive comparison with state-of-the-art encryption techniques, as proposed in ([37–41,44–50]). It is evident that the proposed encryption technique surpasses all previously mentioned encryption techniques when applied to encrypting "Lenna Gray.png". Furthermore, when compared to the existing techniques ([39,41,43–45,48,52]) that have encrypted the "Peppers.png" image, Table 8 provides compelling evidence of the superior performance of the proposed encryption technique across various metrics, including encryption time, MSE, correlation, entropy, UACI, NPCR, BACI, and more. Finally, in Table 9, the comparison results with existing encryption techniques ([37,42–48,52]) that utilized "baboon.tiff", "white.png", and "black.png" images in their experiments are presented. These results further underscore the superiority of the proposed encryption technique over the existing methods.

In sum, the series of experiments conducted in this study consistently demonstrate the clear and substantial superiority of the proposed encryption techniques over existing encryption methods. Across various test scenarios, including the encryption of the "Lenna Gray.png", "Peppers.png", "baboon.tiff", "white.png", and "black.png" images, the proposed encryption technique consistently outperforms its counterparts. This superiority is evident in multiple performance metrics, such as reduced encryption times, a lower MSE, improved correlation, enhanced entropy, and better performance in metrics like UACI, NPCR, and BACI. These compelling results collectively reinforce the effectiveness and robustness of the proposed encryption techniques, making them a noteworthy advancement in the field of image encryption.

Table 6. The results of running CIC1 on "Colored Lenna.png" compared to Ref. [43].

	CIC1	**Ref. [43] (256,256)**
Time in ms	**2.38**	**1670**
seed	0.407910552	
seedp	0.320717184	
seedk	0.238859246	
MSE Original vs. Encrypted		
ch1	7094.403336	
ch2	9068.194191	
ch3	10655.87563	

Table 6. *Cont.*

	CIC1	Ref. [43] (256,256)
Time in ms	**2.38**	**1670**
PSNR		
ch1	9.621644852	
ch2	8.555595491	
ch3	7.854912181	
X^2 Original	236580.67	
X^2 ENC	259.6582	
Correlation of Original		
Horizontal Correlation Coefficient	0.971928377	
Vertical Correlation Coefficient	0.985029604	
Diagonal Correlation Coefficient	0.959331116	
Correlation of Encrypted		
Horizontal Correlation Coefficient	0.002439681	0.0027
Vertical Correlation Coefficient	−0.000653871	−0.0027
Diagonal Correlation Coefficient	−0.004147642	0.0003
Entropy of Original		
Red Channel	6.968426946	
Green Channel	7.594037916	
Blue Channel	7.253102357	
Original vs. Encrypted		
Channel 0 UACI	27.58842019	33.4531
Channel 0 NPCR	99.62539673	99.6078
Channel 1 UACI	30.6128558	
Channel 1 NPCR	99.6181488	
Channel 2 UACI	33.04622276	
Channel 2 NPCR	99.62730408	
Encrypted vs. One-Pixel Modified Encryption		
CH0 UACI	33.45865287	33.4531
CH0 NPCR	99.63378906	99.6078
CH1 UACI	33.44524907	
CH1 NPCR	99.5967865	
CH2 UACI	33.47020018	
CH2 NPCR	99.59068298	
Encrypted vs. Key Modified Encrypted		
CH0 UACI	33.51181329	
CH0 NPCR	99.61051941	
CH1 UACI	33.50390266	
CH1 NPCR	99.61357117	
CH2 UACI	33.51181329	
CH2 NPCR	99.61051941	
BACI Encryption vs. Pixel Modified Encrypted		
CH0	0.267705363	
CH1	0.26795032	
CH2	0.267637681	
BACI Encrypted vs. Key Modified Encrypted		
CH0	0.267943492	
CH1	0.267892948	
CH2	0.267742368	

Table 7. The results of running GIC on "Gray Lenna.png" compared to results from a number of articles.

	GIC	[38] 6_Rounds	[39]	[40] 256 * 256	[41] CPU Size Only	[41] GPU	[44] Single Core	[44] Multi_Cores	[44] GPU	[45] GPU	[37] Gray Avg	[37] Color Avg	[46]	[47]	[48]	[49]	[50]
Time in ms	**1.26**	**6.739**	**1.26**		**103.4**	**11.4**	**0.9126 * 4**	**0.4832 * 4**	**0.0862 * 4**	**3.4365**	**450.7**	**1353.3**	**80 ms**		**402**	**600.7 (256, 256)**	**1100**
seed	0.573094037																
seedp	0.469779666																
seedk	0.442810324																
MSE of Original vs. Encrypted																	
ch1	8087.48698																
PSNR																	
ch1	9.052667663																
X^2 Original	159456.94																
X^2 Encrypted	245.01074												230	265		248.875	
Correlation of Original																	
Horizontal Correlation Coefficient	0.976212614																
Vertical Correlation Coefficient	0.986572767																
Diagonal Correlation Coefficient	0.964138041																
Correlation of Encrypted																	
Horizontal Correlation Coefficient	−0.00117	−0.003	0.0021	−0.0052	0.0149		0.0022			0.0073	0.0019	0.0024	0.00246	−0.002359	0.001	0.0090266	−0.00058
Vertical Correlation Coefficient	−0.001804761	−0.003	0.0019	0.0031	0.0151		0.0015			0.009	0.0036	0.0028	0.00134	0.001385	0.003	−0.0059255	0.0048
Diagonal Correlation Coefficient	−0.00115	−0.009	0.0006	0.0003	0.0113		0.0025			0.0053	0.0033	0.002	0.00132	−0.000238	0.006	0.0055227	−0.0243
Entropy of Original																	
Red Channel	7.457123508	6.43834544			7.9457												
Entropy of encrypted																	
Red Channel	7.999312018	7.819824649		7.9973			7.9991			7.999382	7.9998	7.9997	7.99929	7.99926	7.9994	7.9992856	7.7.9968
Original vs. Encrypted																	
Channel 0 UACI	29.23137141	28.61734577		33.4573	33.28		33.51						33.499	34.048559	33.52	33.4199	33.08
Channel 0 NPCR	99.67632	99.63645935		99.6122	99.65		0.9962						99.616	99.795532	99.63	99.617	99.58
Encrypted vs. one-pixel Modified Encrypted																	
CH0 UACI	33.44539717																
CH0 NPCR	99.631732																
Encrypted vs. Key Modified Encrypted																	
CH0 UACI	33.4						33.45			33.691						33.51	
CH0 NPCR	99.61013794						0.9961			99.6399						99.63	
BACI of Encrypted vs. Pixel Modified Encrypted																	
CH0	0.267822753																
BACI Encrypted vs. Key Modified Encrypted																	
CH0	0.267919803																

Table 8. The results of running CIC1 on "Colored Peppers.tiff" compared to results from a number of articles.

	CIC1	[39]	[41] CPU Size Only	[41] GPU	[43]	[44]	[45]	[48]	[52]
Time in ms	**2.2**	**1.26**	**103.4**	**11.4**	**5130**				**1756**
seed	0.41079142								
seedp	0.323598052								
seedk	0.188286409								
MSE Original vs. Encryption									
ch1	11168.92522								
ch2	11205.73839								
ch3	8015.177773								
PSNR									
ch1	7.650689775								11.89
ch2	7.636398818								
ch3	9.091672016								
X^2 Original	339975.53								
X^2 Encryption	239.7806								
Correlation of Original									
Horizontal Correlation Coefficient	0.976770914								
Vertical Correlation Coefficient	0.97920518								
Diagonal Correlation Coefficient	0.963934813								
Correlation of Encrypted									
Horizontal Correlation Coefficient	0.003734691	0.0021			0.0035	0.0038	0.0092		
Vertical Correlation Coefficient	-2.07E-03	0.0019			-0.0054	0.0007	0.0068		
Diagonal Correlation Coefficient	-1.56E-03	0.0006			0.0012	0.0032	0.0084		
Entropy of Original									
Red Channel	7.058305579								
Green Channel	7.496253345								
Blue Channel	7.338826961								
Entropy of Encrypted									
Red Channel	7.999265701		7.9472		7.9959	7.9998	7.999283		
Green Channel	7.999383295								
Blue Channel	7.999368216								

Table 8. *Cont.*

	CIC1	[39]	[41] CPUsizeonly	[41] GPU	[43]	[44]	[45]	[48]	[52]
Time in ms	**2.2**	**1.26**	**103.4**	**11.4**	**5130**				**1756**
Original vs. Encrypted									
Channel 0 UACI	33.84835337								
Channel 0 NPCR	99.61853027								
Channel 1 UACI	33.87297088								
Channel 1 NPCR	99.60289001								
Channel 2 UACI	29.00593776								
Channel 2 NPCR	99.60365295								
Encrypted vs. One Pixel Modified Enc									
CH0 UACI	33.3822213	38.572	33.5		33.3626	33.48	33.4349	33.45	34.35
CH0 NPCR	99.60517883	99.629	99.85		99.5818	0.9662	99.6244	99.64	99.98
CH1 UACI	33.45076617								
CH1 NPCR	99.60136414								
CH2 UACI	33.45834321								
CH2 NPCR	99.6131897								
Encrypted vs. Key Modified Encrypted									
CH0 UACI	33.42832079					33.37			
CH0 NPCR	99.58992004					0.9958			
CH1 UACI	33.41574575								
CH1 NPCR	99.60479736								
CH2 UACI	33.42832079								
CH2 NPCR	99.58992004								
BACI Encrypted vs. Pixel Modified Encrypted									
CH0	0.267719515								
CH1	0.26752743								
CH2	0.267476458								
BACI Encrypted vs. Key Modified Encrypted									
CH0	0.267598371								
CH1	0.267275973								
CH2	0.267223026								

Table 9. The results of running CIC3 on "Colored Baboon.tiff", "white.png", and "black.png" compared to results from a number of articles

	CIC3: Baboon	CIC3: White	CIC3: Black	[42] Baboon	[42] White	[42] Black	[43] Baboon (256, 256)	[44] Baboon	[45] Baboon	[37] Baboon	[46] Baboon	[47] Baboon	[48] Baboon	[52]
Time in ms	**2.5**	**1.6**	**1.0825** $\times 10^3$	**925 avg**			**1.65** $\times 10^3$							**2.456** $\times 10^3$
seed	0.223152919	0.383265968	0.269633424											
seedp	0.635959551	0.617574674	0.356997044											
seedk	0.278755113	0.494579228	0.575921297											
MSE Original vs. Encrypted														
ch1	9485.679642	21709.18221	21718.32024	6.80×10^3	2.17×10^4	2.18×10^4								
ch2	7743.157845	21761.82545	21714.44178											
ch3	8642.982521	21722.63377	21717.76086											
PSNR														
ch1	8.360119077	4.764368972	4.762541283	9.8067	4.7607	4.7503								13.08
ch2	9.241622484	4.753850382	4.763316915											
ch3	8.764167262	4.761678805	4.762653142											
X^2 Original	100859.73	261120	2.55×10^{10}											
X^2 Encrypted	240.7334	254.85449	258.76398								258	250		
Correlation Original														
Horizontal Correlation Coefficient	0.866543103	1	1											
Vertical Correlation Coefficient	0.758730154	1	1											
Diagonal Correlation Coefficient	0.726188678	1	1											
Correlation of Encrypted														
Horizontal Correlation Coefficient	0.004979382	5.78×10^{-4}	-1.48×10^{-5}				2.00×10^{-4}	2.50×10^{-3}	1.90×10^{-3}			-2.128×10^{-3}	2.00×10^{-3}	
Vertical Correlation Coefficient	−0.004313788	0.000148297	-5.86×10^{-5}				-1.00×10^{-4}	0.0052	0.0074			-1.812×10^{-3}	0.004	
Diagonal Correlation Coefficient	−0.002290213	0.000727736	−0.000299487				0.0027	0.0018	0.0005			-1.473×10^{-3}	0.008	
Entropy of Original														
Red Channel	7.752217172	0	0											
Green Channel	7.474431586	0	0											
Blue Channel	7.706671843	0	0											

Table 9. *Cont.*

	CIC3: Baboon	CIC3: White	CIC3: Black	[42] Baboon	[42] White	[42] Black	[43] Baboon (256, 256)	[44] Baboon	[45] Baboon	[37] Baboon	[46] Baboon	[47] Baboon	[48] Baboon	[52]
Time in ms	**2.5**	**1.6**	**1082.5**	**925avg**			**1650**							**2456**
Entropy of Encrypted														
Red Channel	7.999361961	7.999290305	7.999998056				7.9959	7.9999	7.99928		7.99925	7.99931	7.9993	
Green Channel	7.999348141	7.999319074	7.999998152											
Blue Channel	7.999293719	7.999278883	7.999998185											
Original vs. Encrypted														
Channel 0 UACI	31.25569362	50.00645656	50.00223729	33.54939	33.54164	33.53024	33.3373	33.42	33.4354	33.47		33.62068		26.67
Channel 0 NPCR	99.57885742	99.59144592	99.610097	99.59793	99.60442	99.61929	99.6246	0.996	99.6132	99.63		99.666976		99.98
Channel 1 UACI	28.58791576	50.07205739	49.99562481											
Channel 1 NPCR	99.62654114	99.59564209	99.608358											
Channel 2 UACI	29.98062732	50.02933278	50.00090204											
Channel 2 NPCR	99.62844849	99.60479736	99.609502											
Encrypted vs. One-pixel Modified Encrypted														
CH0 UACI	33.4608639	33.39831782	33.46571685					33.51						
CH0 NPCR	99.609375	99.62501526	99.608265					0.9961						
CH1 UACI	33.55413998	33.43291339	33.46277882											
CH1 NPCR	99.62005615	99.60670471	99.608928											
CH2 UACI	33.467678	33.47475388	33.46288022											
CH2 NPCR	99.61357117	99.60670471	99.609083											
Encrypted vs. Key Modified Encrypted														
CH0 UACI	33.46980974	33.55993233	33.46169072											
CH0 NPCR	99.60289001	99.62539673	99.608933											
CH1 UACI	33.43826593	33.51012286	33.46330268											
CH1 NPCR	99.62043762	99.61128235	99.61017											
CH2 UACI	33.46980974	33.55993233	33.46304682											
CH2 NPCR	99.60289001	99.62539673	99.617289											

Table 9. *Cont.*

	CIC3: Baboon	CIC3: White	CIC3: Black	[42] Baboon	[42] White	[42] Black	[43] Baboon (256, 256)	[44] Baboon	[45] Baboon	[37] Baboon	[46] Baboon	[47] Baboon	[48] Baboon	[52]
Time in ms	**2.5**	**1.6**	**1082.5**	**925avg**			**1650**							**2456**
BACI Encrypted vs. Pixel Modified Encrypted														
CH0	0.267642274	0.267526324	0.267686784											
CH1	0.267546125	0.267869938	0.267712745											
CH2	0.267424658	0.267902475	0.267707479											
BACI Encrypted vs. Key Modified Encrypted														
CH0	0.268003697	0.267389723	0.267668858											
CH1	0.267900605	0.268049608	0.267694917											
CH2	0.267638505	0.267642845	0.267700614											

4.2.10. Time Complexity and Average Time

The time complexity of all six algorithms is linear, denoted as O(n), concerning the number of pixels in the image. For grayscale images, we generated only $2 * r$ and $2 * c$ random numbers. Generating a random number with SBTM or SBTMPi was constant. Subsequently, each pixel was encrypted according to step 11 of GIC, wherein all the calculations were performed in constant time. Thus, the algorithm exhibits a linear time complexity relative to the number of pixels. The average encryption time is of significant importance, albeit challenging to directly compare with other algorithms due to variations in hardware, images, and running environments. Nonetheless, it is crucial to emphasize that the proposed algorithms showcased excellent performance with the utilized hardware. Notably, encrypting a colored Lenna image of size (512, 512) took a mere 1.26 ms, while encrypting the considerably larger image Black.png (100 MB) only required 1.0825 seconds. The decryption results are even more impressive since the decryption algorithm does not need to calculate the digest, *ES*, leading to an even faster processing times.

Figure 9 shows an image in the following forms: plain, encrypted, one pixel changed, key change of 10^{-15}, both pixel and key changes, and decrypted.

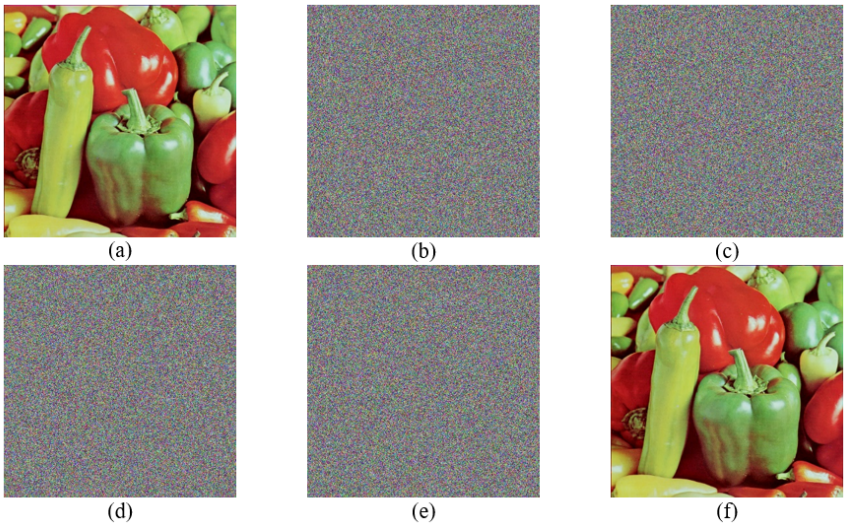


Figure 9. Result of MODPI on Peppers.tiff: (**a**). Plain, (**b**). Encrypted, (**c**). Encrypted/one pixel changed, (**d**). Encrypted/ key change of 10^{-15}, (**e**). Encrypted/both pixel and key changes, and (**f**). Decrypted

4.2.11. Exploring the Performance and Attributes of Proposed Ciphers

In this section, we explore the attributes claimed for the proposed ciphers, namely fast, strong, and parallel. The encryption time achieved using the proposed ciphers is remarkably short compared to the existing literature, as evidenced in Table 7. Our cipher was able to encrypt the image Lenna in just 1.26 ms, whereas the reported literature ranges from 1.26 ms to over 1300 ms. Similar outcomes are observed in Tables 8 and 9. Notably, our ciphers required only 1082.5 ms to encrypt the Black image, which consists of 100 million pixels. This exceptional speed led us to consider our ciphers to be fast. Assessing the strength of an encrypted image involves employing various statistical tests. In Sections 4.2.1–4.2.5, we discuss a comprehensive array of tests conducted on the encrypted images. The results closely align with the theoretical values for each test, indicating that our ciphers possess a remarkable strength, thus earning the label of strong. Our implementation prioritized parallel processing to enhance efficiency. We found no practical limit to the number of sub-images we could create, which translates to no practical limit to the number of threads operating on an image. Figure 10 illustrates this, demonstrating a nearly tenfold speedup when dividing the image into 30 sub-images compared to just one sub-image (one thread on CIC1). Although utilizing more sub-images is feasible, the overhead of managing threads

becomes significant, especially considering our use of a single CPU with only 14 cores. For larger images (24 million pixels) or more, we experimented with 100 sub-images, which utilized 300 threads on CIC3, and we still observed improvements in the encryption time, as depicted in Figure 8. Therefore, we confidently affirm that the proposed ciphers utilize parallel execution to improve the encryption time by a good margin.

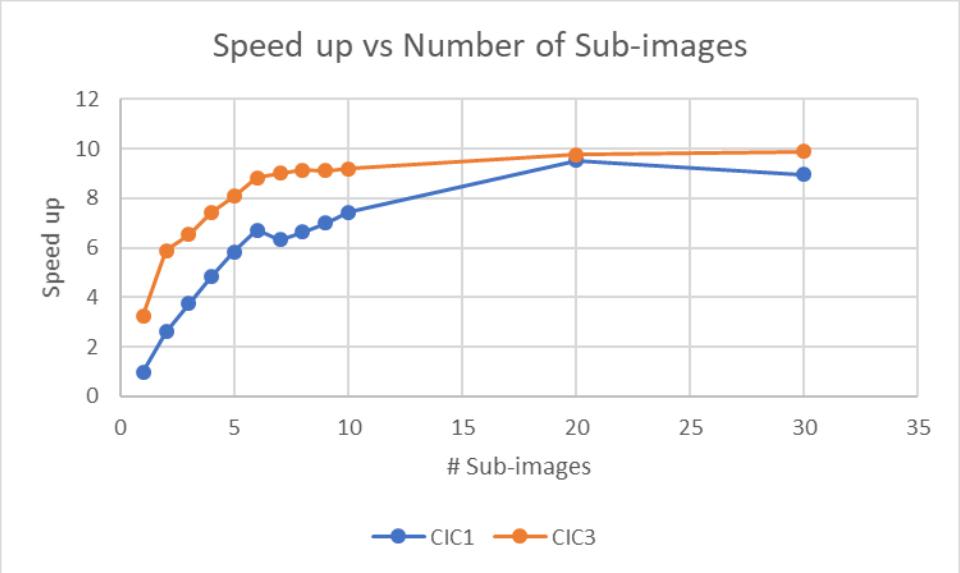


Figure 10. Speedup: This figure shows that a speedup of nearly 10 is obtained as the number of threads increases from 1 to [30 in CIC1, 90 in CIC3] to encrypt a 24M-Pixel colorful image. Note that the number of cores is constant (14 cores).

5. Conclusions

This paper presents a significant contribution to the field of image encryption through the development of a novel, fast, and strong cryptographic scheme. The proposed colored image cipher, built upon a state-based modified tent map and cryptographically secure pseudo-random number generators, addresses several key challenges in image encryption and security. One of the primary achievements of this work is the introduction of a parallel implementation approach. By leveraging parallel processing techniques, the encryption process is expedited, enabling the real-time encryption and decryption of colored images. This advancement is particularly relevant in today's fast-paced digital environment, where speed and efficiency are of utmost importance. The paper also introduces a novel approach to confusion and diffusion, essential components of effective encryption algorithms. By incorporating state-based modified tent map CPRNGs, the proposed cipher achieves an enhanced level of cryptographic strength. The synergy between the CPRNG and the encryption algorithm results in a robust system that withstands various cryptanalysis techniques.

Furthermore, this research tackles the crucial aspect of random number generation. The employed cryptographically secure pseudo-random number generator ensures a high degree of randomness, making the generated keys and masks resilient to prediction and attacks. This strong foundation contributes to the overall security of the proposed image cipher. Efficiency in encryption algorithms is another hallmark of this work. The developed encryption scheme is tailored to address the unique characteristics of colored images, optimizing both security and computational efficiency. This practicality ensures that the proposed cipher is not only theoretically robust but also feasible for real-world applications. Lastly, the cipher exhibits a remarkable sensitivity to key variations and plane image modifications. This attribute enhances the security of the encrypted images by rendering unauthorized decryption attempts ineffective. The proposed scheme's robustness against unauthorized access is a testament to its practicality in safeguarding sensitive visual infor-

mation. In sum, this paper's contributions mark a significant step forward in the realm of image encryption. The combination of a highly parallel implementation, a novel confusion and diffusion strategy, a strong CPRNG, efficient encryption algorithms for colored images, and heightened sensitivity to key and plane image modifications collectively form a comprehensive and effective image encryption scheme. As digital data security becomes increasingly paramount, the findings presented in this paper offer valuable insights and methodologies for advancing the field of image encryption and secure data communication.

Future work should encompass an exploration of adaptive encryption algorithms, an investigation of hardware implementation, an integration with cloud services, a consideration of post-quantum security, and user experience enhancement. Additionally, potential extensions to multi-modal encryption, the exploration of advanced parallelization techniques, contributions to cryptographic standardization efforts, and real-world deployment scenarios are vital directions to further strengthen and apply the proposed image cipher's contributions.

Author Contributions: Conceptualization: A.A.-D. and Y.S.; methodology: A.A.-D., Y.S., S.F. and S.A.-E.; formal analysis: A.A.-D. and Y.S.; investigation: S.F. and S.A.-E.; writing—original draft preparation: A.A.-D. and Y.S.; writing—review and editing: S.F. and S.A.-E.; supervision: A.A.-D. and Y.S.; funding acquisition: S.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Artificial Intelligence Research Center (AIRC), College of Engineering and Information Technology, Ajman University, Ajman, UAE.

Data Availability Statement: Links to sources of the images used in our experimentation: 1. Created Images (White, Black and Gray Lenna): https://drive.google.com/drive/folders/18o9786iPDNmijDx4Ze6dCV24mWqRChVj?usp=drive_link; 2. Colored Lenna: https://en.wikipedia.org/wiki/Lenna; 3. Peppers and Baboon: https://sipi.usc.edu/database/; 4. Images from unsplash.com: https://unsplash.com/photos/woman-checking-labels-nHRXNv2qeDE, https://unsplash.com/photos/grayscale-photo-of-short-coated-dog-M7l0CS4yBsY, https://unsplash.com/photos/white-and-black-flower-in-white-background-ybDvbCvh9Ro, (Sources were accessed on 2 May 2024).

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

Appendix A. GIC Detailed Implementation

GIC introduces a groundbreaking strategy into the confusion phase, optimizing parallelism to its fullest potential. Additionally, it incorporates another innovative technique in the diffusion phase, enhancing efficiency and sensitivity to variations. A detailed description of this algorithm is provided below:

1. Load grayscale image.
2. Get image dimensions: r and c.
3. Allocate matrices and arrays: one matrix, I, size $r * c$, to store the original image, another matrix, E, size $r * c$, to store the encrypted image, two arrays, $Roll1$ and $Roll2$, of lengths r and c of integers to store shift values, and finally, two arrays, $Diff1$ and $Diff2$, of lengths r and c of doubles to store values in order to assist in the diffusion process.
4. Initialize variables: GIC uses six keys $(uk1 \ldots uk6)$, a user-provided seed, US, of the double type, and two integer values, R and C, to divide the image, all read from the user; a fixed initial seed, IS, holds the fraction part of PI.
5. Extract the pixel values into matrix I.
6. Divide the image: generate indices for $R * C$ sub-matrices. For example, if $R = 2$, $C = 2$, $r = 6$, and $c = 7$, the generated indices of the four sub-matrices would be as follows: $sub1 = [0, 0, 3, 3]$, $sub2 = [0, 3, 3, 7]$, $sub3 = [3, 0, 6, 3]$, and $sub4 = [3, 3, 6, 7]$. Notice that, if r does not divide R or if c does not divide C without a remainder, the extra pixels are added to the right and bottom sub-matrices.
7. Generate encryption keys: use the user-provided keys: $uk1 \ldots uk6$ and US to generate two encryption keys, $ek1$ and $ek2$, as follows:

(a) Initialize an SBTM object with $(seed = IS/2 + US/2)$ as a seed and $uk1$ for the control variable.
(b) Generate n random numbers; a value of 97 is used for n. Note that SBTM is extremely chaotic.
(c) Set the control variable of SBTM to $uk2$; this changes the control variable but keeps the state of the STBM object intact.
(d) Repeat steps b and c for $uk3 \ldots uk6$. At this point, the internal state of SBTM is influenced by all six user keys.
(e) Generate $ek1$ and $ek2$: use the next double random number, $n1$, and the 97th random number $n97$ to generate $ek1 = n1$ and $ek2 = n97$.

8. Generate $Diff1$ and $Diff2$: create two threads, one to generate random double values for $Diff1$ and the other to generate random values for $Diff2$. The SBTM used is initialized with $seed = IS/2 + US/2$, and the control variable is $ek1$ and $ek2$, respectively.
9. Generate the digest, ES: the digest should be sensitive to any change in the original image. i.e., a change of one to any pixel of the image should produce a significantly different ES. The extremely chaotic and random behavior of SBTM is used to produce a strong and efficient digest as follows: create $R * C$ threads, one for each sub-image/sub-matrix; each thread returns $Seed_{\mathrm{k}}$ as follows:

$Seed_{\mathrm{k}} \leftarrow 0$
$i \leftarrow sub_{\mathrm{k}}[0]$
while i < sub$_{\mathrm{k}}$ [2] **do**
  $j \leftarrow sub_{\mathrm{k}}[1]$
  while j < sub$_{\mathrm{k}}$ [3] **do**
    $Seed_{\mathrm{k}} \leftarrow Seed_{\mathrm{k}} + (I[i][j] + Diff2[j]) * (Diff1[i] + Diff2[j])$
    $Seed_{\mathrm{k}} \leftarrow$ Fraction part of $Seed_{\mathrm{k}}$
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while
return $Seed_{\mathrm{k}}$

at this point, we have k sub-seeds $Seed_{\mathrm{k}}$. Calculate ES as follows:

$$ES \leftarrow \sum_{1}^{k} Seed_{\mathrm{k}}$$

$$ES \leftarrow ES - (int)ES$$

$$seed \leftarrow seed/2 + ES/2$$

ES is a very random digest influenced by all user variables and all pixel values of the image, I. Save ES to file. At this point, two strong keys and a strong seed are created, and the encryption process can commence.

10. Generate $Roll1$ and $Roll2$: create two threads, one to generate random integer values for $Roll1$ and the other to generate random integer values for $Roll2$. The SBTM used is initialized with $seed$, and the control variable is $ek1$ and $ek2$, respectively. Note that $Roll1$ and $Roll2$ hold unique random values in the ranges of [0, r] and [0, c], respectively.
11. Encrypt the image: create $R * C$ threads, one for each $sub - image/sub - matrix$. Each thread will do the following, assuming it will work on sub$_{\mathrm{k}}$:

$i \leftarrow sub_{\mathrm{k}}[0]$
while i < sub$_{\mathrm{k}}$ [2] **do**
  $j \leftarrow sub_{\mathrm{k}}[1]$
  while j < sub$_{\mathrm{k}}$ [3] **do**
    Simulate the shifting of pixels as follows:

Calculate new row value as $row \leftarrow (i + Roll2[j])\%r$
Calculate new column value as $col \leftarrow (j + Roll1[row])\%c$
(This will give us the coordinates of a $I[i, j]$ after being shifted/rolled horizontally and vertically without actually moving the pixel value to this new location. This gives a huge performance boost and allows for the parallel encryption of the image.)
$x \leftarrow Diff1[row] * 256$
$y \leftarrow Diff2[col] * 256$
Encrypt pixel
$E[i, j]$ as: $E[i, j] \leftarrow I[row, col] \bigoplus((Roll1[i] \bigoplus Roll2[j])\%256)) \bigoplus x \bigoplus y$
(This step is another performance improvement. Usually, a random number is generated and XORed with $I[row, col]$. Here, however, the random integers used to confuse the image are also used to encrypt it. An XOR and a modulus replace a function call.)
$j \leftarrow j + 1$
end while
$i \leftarrow i + 1$
end while

It is very important to understand that all threads are working on separate regions of the shared matrix, E, and hence, no locking or synchronization is needed.

12. Create an image, *encImage*, from E.
13. Save *encImage* to file.
14. Save the six user keys $uk1$ through 6, and the user seed US, or share them with people who will have access to decrypt the image.

Appendix B. CIC1 Detailed Implementation

This version was developed to encrypt colored images. It uses one thread to encrypt all three channels of a sub-image. It follows the same procedure as GIC. The differences are highlighted below:

1. Load colored image.
2. Get image dimensions: r and c.
3. Allocate matrices and arrays: three matrices, IR, IG, and IB, size $r * c$, to store the original image, another three matrices, ER, EG, and EB, size $r * c$, to store the encrypted image, three pairs of arrays, $Roll1$ and $Roll2$ of length r and c of integers to store shift values ($Roll1r$ and $Roll2r$ are used for the red channel, $Roll1g$ and $Roll2g$ for the green channel, and $Roll1b$ and $Roll2b$ for the blue one), and three pairs of arrays, $Diff1$ and $Diff2$ of length r and c of doubles to store values to assist in the diffusion process ($Diff1r$ and $Diff2r$ are used for the red channel, $Diff1g$ and $Diff2g$ for the green channel, and $Diff1b$ and $Diff2b$ for the blue one).
4. Initialize variables: same as step 4 in GIC.
5. Extract pixel values into matrices IR, IG, and IB
6. Divide the image: same as 6 in GIC.
7. Generate encryption keys: use the user-provided keys and seed, $uk1 \ldots uk6$, and US to generate six encryption keys $ek1$ through $ek6$ as follows:
   a. Initialize an SBTM object with $(seed = IS/2 + US/2)$ as a *seed* and $uk1$ for the control variable
   b. Generate n random numbers; a value of 97 is used for n. (Note that SBTM is extremely chaotic.)
   c. Set the control variable of SBTM to $uk2$; (this changes the control variable but keeps the state of the STBM object intact.)
   d. Repeat b and c for $uk3 \ldots uk6$. At this point, the internal state of SBTM is influenced by all six user keys.
   e. Generate $ek1$ through $ek6$:
   $j \leftarrow 1$
   while j <= 6 **do**

$ekj \leftarrow$ next random number
generate 97 random numbers (note that 97 is a prime number. Any arbitrary number of steps can be used as SBTM produces drastically different next values.)
$j \leftarrow j + 1$
end while

8. Generate six Diff arrays: create six threads; each one generates random double values for one of the Diff arrays. The SBTM objects used are initialized with $seed = IS/2 + US/2$ and one of the six keys, $ek1 \ldots ek6$, for its control variable.
9. Generate the digest, ES: create $R * C$ threads to calculate k sub-seeds, one for each sub-image. Then, calculate ES as follows:

$Seed_{\mathrm{k}} \leftarrow 0$
$i \leftarrow sub_{\mathrm{k}}[0]$
while i < sub$_{\mathrm{k}}$ [2] **do**
$j \leftarrow sub_{\mathrm{k}}[1]$
while j < sub$_{\mathrm{k}}$ [3] **do**
$Seed_{\mathrm{k}} \leftarrow Seed_{\mathrm{k}} + (IR[i][j] + Diff2r[j]) * (Diff1r[i] + Diff2r[j])$
$Seed_{\mathrm{k}} \leftarrow$ Fraction part of $Seed_{\mathrm{k}}$
$Seed_{\mathrm{k}} \leftarrow Seed_{\mathrm{k}} + (IG[i][j] + Diff2g[j]) * (Diff1g[i] + Diff2g[j])$
$Seed_{\mathrm{k}} \leftarrow$ Fraction part of $Seed_{\mathrm{k}}$
$Seed_{\mathrm{k}} \leftarrow Seed_{\mathrm{k}} + (IB[i][j] + Diff2b[j]) * (Diff1b[i] + Diff2b[j])$
$Seed_{\mathrm{k}} \leftarrow$ Fraction part of $Seed_{\mathrm{k}}$
$j \leftarrow j + 1$
end while
$i \leftarrow i + 1$
end while
return $Seed_{\mathrm{k}}$
Follow the rest of step 9 in GIC.

ES is a very random digest influenced by all user variables and all pixel values of image I. Save ES to file. At this point, six strong keys and a strong seed are created, and the encryption process can commence.

10. Generate six roll arrays: create six threads; each one generates random integer values for one of the Roll arrays. The SBTM objects used are initialized with *seed* and one of the six keys, $ek1 \ldots ek6$, for its control variable. Note that $Roll1$ and $Roll2$ arrays hold unique random values in the ranges of [0, r] and [0, c], respectively.
11. Encrypt the image: create $R * C$ threads, one for each sub-image. Each thread will do the following, assuming it will work on sub$_{\mathrm{k}}$:

$i \leftarrow sub_{\mathrm{k}}[0]$
while i < sub$_{\mathrm{k}}$ [2] **do**
$j \leftarrow sub_{\mathrm{k}}[1]$
while j < sub$_{\mathrm{k}}$ [3] **do**
Simulate the shifting of pixels as follows:
For the red channel:
1. Calculate the new row value as follows: $row \leftarrow (i + Roll2r[j])\%r$
2. Calculate the new column value as follows: $col \leftarrow (j + Roll1r[row])\%c$
3. $x \leftarrow Diff1r[row] * 256$
4. $y \leftarrow Diff2r[col] * 256$
5. Encrypt pixel $ER[i, j]$ as follows:
$E[i, j] \leftarrow IR[row, col] \bigoplus ((Roll1r[i] \bigoplus Roll2r[j])\%256)) \bigoplus x \bigoplus y$
6. Repeat 1, 2, 3, 4, and 5 using IG, EG, $Roll1g$, $Roll2g$, $Diff1g$, and $Diff2g$.
7. Repeat 1, 2, 3, 4, and 5 using IB, EB, $Roll1b$, $Roll2b$, $Diff1b$, and $Diff2b$.
$j \leftarrow j + 1$
end while
$i \leftarrow i + 1$
end while

(Note that this thread encrypted all the pixels from the three channels of this sub-image)

12. Create an image, *encImage*, from matrices ER, EG, and EB.
13. Save *encImage* to file.
14. Save/share user keys and US.

Appendix C. CIC3 Detailed Implementation

This version is a very similar to CIC1; however, in step 11, a thread working on sub_k creates three threads, one to encrypt the red channel of this sub-image and the other two to encrypt the green and blue channels. So, for example, if $R = 2$ and $C = 2$, four sub-images will be created. While CIC1 will create only four threads to do the encryption step (11), CIC3 will create 12 threads to do the same step.

Appendix D. Decryption Detailed Implementation

Appendix D.1. GICD

The description of the decryption algorithm is given below:

1. Load ES from file, and then load the encrypted image.
2. Get image dimensions: r and c.
3. Allocate matrices and arrays: one matrix, E, size $r * c$, to store the encrypted image, another matrix, D, size $r * c$, to store the decrypted image, and two arrays, $Roll1$ and $Roll2$, of lengths r and c of integers to store shift values. Finally, two arrays, $Diff1$ and $Diff2$, of lengths r and c of doubles to store values are used to assist in the diffusion process.
4. Initialize variables: GICD uses six keys $(uk1 \ldots uk6)$, the user seed, US, digest ES of the double type generated in the encryption part, and two integer values, R and C, to divide the image, all read from the user. Providing different values for R and C does not affect the process.
5. Extract pixel values into matrix E.
6. Divide the image: generate indices for the $R * C$ submatrices. Same as step 6 in GIC.
7. Generate encryption keys: use the user-provided keys $uk1 \ldots uk6$, and US to generate two encryption keys, $ek1$ and $ek2$, as in step 7 of GIC.
8. Generate Diff arrays: same as step 8 in GIC.
9. Generate $Roll1$ and $Roll2$: same as step 10 in GIC; use $seed = ES/2 + seed/2$ as a seed for SBTM.
10. Decrypt the image: create $R * C$ threads, one for each sub-image. Each thread will do the following, assuming it will work on sub_k:

    $i \leftarrow sub_\text{k}[0]$
    while i < sub_k [2] **do**
        $j \leftarrow sub_\text{k}[1]$
        while j < sub_k [3] **do**
            Simulate the shifting of pixels as follows:
            Calculate new row value as follows: $row \leftarrow (i + Roll2[j])\%r$
            Calculate new column value as follows: $col \leftarrow (j + Roll1[row])\%c$
            $x \leftarrow Diff1[row] * 256$
            $y \leftarrow Diff2[col] * 256$
            Decrypt pixel
    $D[row, col]$ as follows: $D[row, col] \leftarrow E[i, j] \bigoplus y \bigoplus x((Roll1[i] \bigoplus Roll2[j])\%256))$
            $j \leftarrow j + 1$
        end while
        $i \leftarrow i + 1$
    end while

11. Create an image, *DecImage*, from matrix D.

12. Save *DecImage* to disk. A lossless format such as PNG is used for the ability to compare it with the original image.

Appendix D.2. CIC1D

1. Load *ES* from file, then load the encrypted color image.
2. Get image dimensions: r and c.
3. Allocate matrices and arrays: three matrices, DR, DG, and DB, size $r * c$, to store the decrypted image, another three matrices, ER, EG, and EB, size $r * c$, to store the encrypted image, three pairs of arrays, $Roll1$ and $Roll2$ of lengths r and c of integers to store shift values ($Roll1r$ and $Roll2r$ are used for the red channel, $Roll1g$ and $Roll2g$ for the green channel, and $Roll1b$ and $Roll2b$ for the blue one), and three pairs of arrays, $Diff1$ and $Diff2$ of lengths r and c of doubles to store values to assist in the diffusion process ($Diff1r$ and $Diff2r$ are used for the red channel, $Diff1g$ and $Diff2g$ for the green channel, and $Diff1b$ and $Diff2b$ for the blue one).
4. Initialize variables: same as step 4 in GICD.
5. Extract pixel values into matrices ER, EG, and EB
6. Divide the image: same as in step 6 in GICD. Providing different values for R and C does not affect the process.
7. Generate encryption keys: use the user-provided keys and seed, $uk1 \ldots uk6$, and US to generate six encryption keys, $ek1$ through $ek6$, as in step 7 of CIC1.
8. Generate six Diff arrays: same as step 8 in CIC1. use $seed = IS/2 + US/2$ to initialize SBTM.
9. Generate six Roll arrays: create six threads; each one generates random integer values for one of the Roll arrays. The SBTM objects used are initialized with $seed = ES/2 + seed/2$ and one of the six keys, $ek1 \ldots ek6$, for its control variable.
10. Decrypt the image: create $R * C$ threads, one for each sub-image. Each thread will do the following, assuming it will work on sub_k:

    $i \leftarrow sub_\text{k}[0]$
    while i < sub_k [2] **do**
        $j \leftarrow sub_\text{k}[1]$
        while j < sub_k [3] **do**
            Simulate the shifting of pixels as follows:
            For the red channel:
            1. Calculate the new row value as follows: $row \leftarrow (i + Roll2r[j])\%r$
            2. Calculate the new column value as follows: $col \leftarrow (j + Roll1r[row])\%c$
            3. $x \leftarrow Diff1r[row] * 256$
            4. $y \leftarrow Diff2r[col] * 256$
            5. Decrypt pixel $DR[row, col]$ as follows:
    $DR[row, col] \leftarrow ER[i, j] \bigoplus ((Roll1r[i] \bigoplus Roll2r[j])\%256)) \bigoplus x \bigoplus y$
            6. Repeat 1, 2, 3, 4, and 5 using DG, EG, $Roll1g$, $Roll2g$, $Diff1g$, and $Diff2g$.
            7. Repeat 1, 2, 3, 4, and 5 using DB, EB, $Roll1b$, $Roll2b$, $Diff1b$, and $Diff2b$.
            $j \leftarrow j + 1$
        end while
        $i \leftarrow i + 1$
    end while
    (Note that this thread encrypted all the pixels from the three channels of this sub-image)
11. Create an image, *decImage*, from matrices DR, DG, and DB.
12. Save *decImage* to disk. A lossless format such as PNG is used for the ability to compare it with the original image.

Appendix D.3. CIC3D

This version is very similar to CIC1D; however, in step 10, a thread working on sub_k creates three threads, one to decrypt the red channel of this sub-image and the other two to decrypt the green and blue channels. So, for example, if $R = 2$ and $C = 2$, four sub-images will be created. While CIC1 will create only four threads to do the decryption step (10), CIC3D will create 12 threads to do the same step. **GICPiD**, **CIC1PiD**, and **CIC3PiD** use the proposed CPRNG, namely **SBTMPi**.

Appendix D.4. Example

Figure A1 demonstrates how GIC works in a simple example. In this figure, an 8 * 8 image, I, is divided into four sub-images ($Sub0$ to $Sub3$). First, the ES calculation is explained: Four threads are created. Assuming that $Thread0$ will work on $Sub0$, it will work on $Seed0$, initially zero. Considering the pixel at I[0,0], $Seed0 = Seed0 + (I[0,0] + Diff2[0]) * (Diff1[0] + Diff2[0])$. In numbers, $Seed0 = Seed0 + (152 + 0.431) * (0.113 + 0.431) = 82.922$. The fraction part is kept, so $Seed0 = 0.922$. Then, the pixel at I[0,1] is considered, and so on. $Thread1$ will work on $sub1$, starting at I[0,4]. $Thread2$ will work on $sub2$, starting at I[4,0]. Finally, $Thread3$ will work on $sub3$, starting at I[4,4]. Secondly, the encryption process follows. The encryption process is very simple, and it can be represented in the following steps:

$row \leftarrow (i + Roll2[j])\%r$
$col \leftarrow (j + Roll1[row])\%c$
$x \leftarrow Diff1[row] * 256$
$y \leftarrow Diff2[col] * 256$
$E[i,j] \leftarrow I[row, col] \oplus ((Roll1[i] \oplus Roll2[j])\%256)) \oplus x \oplus y$

E is a matrix that holds the encrypted image. Four threads are created. Each thread works on a different sub-image. To show how $E[0,0]$ is encrypted, we need to calculate a new row and column as $row = (0 + 5)\%8 = 5$ and $col = (0, 1)\%8 = 1$. This means I[5,1] will be encrypted and stored in E[0,0]. We then calculate x and y as follows: $x = Diff1[5] * 256 = 0.918 * 256 = 235$ and $y = Diff2[1] * 256 = 0.747 * 256 = 191$. Next, we encrypt E[0,0] as follows: $E[0,0] = I[5,1] \oplus ((2 \oplus 5)\%256)) \oplus 234 \oplus 191 = 80$.

			c	0	1	2	3	4	5	6	7
			Diff2	0.431	0.747	0.742	0.133	0.821	0.034	0.495	0.965
			Roll2	5	7	2	0	4	6	3	1
r	Diff1	Roll1									
0	0.113	2		152	4	148	78	88	50	161	116
1	0.913	7		205	54	Sub 0 165	160	110	57	Sub 1 201	38
2	0.977	0		5	71	125	160	144	219	50	254
3	0.674	4		37	236	133	148	111	94	130	107
4	0.346	6		85	52	4	69	147	242	254	3
5	0.918	1		159	2	Sub 2 241	99	83	123	Sub 3 254	255
6	0.125	3		237	151	133	38	76	91	151	217
7	0.05	5		222	226	41	91	33	24	111	65

Figure A1. Simplified example of 8 * 8 image along with created arrays of random values.

Appendix E. Sensitivity Results

Table A1. Sensitivity to change.

	Lenna.png	Baboon.tiff	Bernard-hermant-nHRXNv2qeDE-unsplash.jpg https	White.png	Black.png	Peppers.tiff	Jocelyn–Morales	Danny	Lennag.png
SBTM-Based Cipher									
Original Image vs. Encrypted Image									
Channel 0 UACI	27.58842019	31.25569362	34.21017768	50.00645656	50.00223729	33.84835337	49.00735754	34.09528189	29.23137141
Channel 0 NPCR	99.62539673	99.57885742	99.6115375	99.59144592	99.610097	99.61853027	99.61275572	99.60907154	9.96×10^1
Channel 1 UACI	30.6128558	28.58791576	33.77552621	50.07205739	49.99562481	33.87297088			
Channel 1 NPCR	99.6181488	99.62654114	99.60817083	99.59564209	99.608358	99.60289001			
Channel 2 UACI	33.04622276	29.98062732	32.76111791	50.02933278	50.00090204	29.00593776			
Channel 2 NPCR	99.62730408	99.62844849	99.61207917	99.60479736	99.609502	99.60365295			
Encrypted Image vs. One-pixel Modified Encrypted Image									
CH0 UACI	33.45865287	33.4608639	33.46644977	33.39831782	33.46571685	33.3822213	33.46162264	33.47240301	33.44539717
CH0 NPCR	99.63378906	99.609375	99.60886667	99.62501526	99.608265	99.60517883	9.96×10^1	9.96×10^1	9.96×10^1
CH1 UACI	33.44524907	33.55413998	33.46451575	33.43291339	33.46277882	33.45076617			
CH1 NPCR	99.5967865	99.62005615	99.60915833	99.60670471	99.608928	99.60136414			
CH2 UACI	33.47020018	33.467678	33.4594643	33.47475388	33.46288022	33.45834321			
CH2 NPCR	99.59068298	99.61357117	99.60969167	99.60670471	99.609083	99.6131897			
Encrypted Image vs. Encrypted Image with 1 bit Modified Key									
CH0 UACI	33.51181329	33.46980974	33.47242631	33.55993233	33.46169072	33.42832079	3.35×10^1	3.35×10^1	3.34×10^1
CH0 NPCR	99.61051941	99.60289001	99.61035833	99.62539673	99.608933	99.58992004	99.61118259	99.60820217	99.61013794
CH1 UACI	33.50390266	33.43826593	33.46260324	33.51012286	33.46330268	33.41574575			
CH1 NPCR	99.61357117	99.62043762	99.607475	99.61128235	99.61017	99.60479736			
CH2 UACI	33.51181329	33.46980974	33.47242631	33.55993233	33.46304682	33.42832079			
CH2 NPCR	99.61051941	99.60289001	99.61035833	99.62539673	99.617289	99.58992004			
Original Image vs. Encrypted Image									
Channel 0 UACI	27.65708175	33.80914127	34.20732696	49.99485988	49.85856898	33.85631486	48.9874037	34.09150993	29.2967643
Channel 0 NPCR	99.62501526	99.59335327	99.6116875	99.62654114	99.60250854	99.60594177	99.61094057	99.60797253	99.61013794
Channel 1 UACI	30.59006036	33.95360909	33.77049199	50.03437416	49.97832654	34.00023516			
Channel 1 NPCR	99.59716797	99.62272644	99.60960417	99.59869385	99.61585999	99.60365295			
Channel 2 UACI	33.04504694	29.0490603	32.76702029	49.97035905	49.99389948	29.0110719			
Channel 2 NPCR	99.6055603	99.62615967	99.6105125	99.61166382	99.60594177	99.6257782			

Table A1. *Cont.*

	Lenna.png	Baboon.tiff	Bernard-hermant-nHRXNv2qeDE-unsplash.jpg https	White.png	Black.png	Peppers.tiff	Jocelyn–Morales	Danny	Lennag.png
SBTMPi-Based Cipher									
Encrypted Image vs. One-pixel Modified Encrypted Image									
CH0 UACI	33.44363942	33.44716689	33.46292327	33.44132666	33.42157401	33.47747504	33.46663227	33.45831224	33.48127926
CH0 NPCR	99.61700439	99.62310791	99.60818333	99.60136414	99.62882996	99.61395264	99.61052913	99.60846462	99.59793091
CH1 UACI	33.45829384	33.44710107	33.46661425	33.47128775	33.50682576	33.49143533			
CH1 NPCR	99.6131897	99.59602356	99.60992083	99.61204529	99.62272644	99.5967865			
CH2 UACI	33.43248405	33.42796924	33.45198296	33.47009547	33.41540318	33.45033683			
CH2 NPCR	99.60021973	99.62730408	99.60812083	99.5967865	99.61624146	99.59907532			
Encrypted Image vs. Encrypted Image with 1 bit Modified Key									
CH0 UACI	33.57714485	33.44705918	33.46863444	33.4321325	33.4786718	33.39455997	33.47517681	33.45534616	33.56725506
CH0 NPCR	99.62501526	99.60479736	99.60951667	99.60632324	99.61433411	99.61433411	99.60845986	99.60848103	99.5967865
CH1 UACI	33.48337211	33.44716988	33.46864886	33.46274881	33.47088982	33.55970794			
CH1 NPCR	99.61967468	99.60021973	99.6104	99.60365295	99.61051941	99.62844849			
CH2 UACI	33.57714485	33.44705918	33.46863444	33.4321325	33.4786718	33.39455997			
CH2 NPCR	99.62501526	99.60479736	99.60951667	99.60632324	99.61433411	99.61433411			
BACI: Encrypted Image vs. One Pixel Modified Encrypted Image									
CH0	0.267672155	0.267825962	0.267701983	0.268154275	0.266954088	0.267924489	0.267713061	0.26774138	0.268127337
CH1	0.266889922	0.267832658	0.267613922	0.267658944	0.267743993	0.267803635			
CH2	0.267346784	0.267305749	0.267742205	0.267814098	0.267509664	0.267944045			
BACI: Encrypted Image vs. Encrypted Image with 1 bit Modified Key									
CH0	0.267588754	0.268173128	0.267719424	0.267722194	0.267755487	0.268633796	0.26767393	0.267654332	0.268329708
CH1	0.268129267	0.267638943	0.267753272	0.267616856	0.267014916	0.26709827			
CH2	0.267633281	0.267697571	0.267743322	0.268080133	0.267898144	0.267849135			

Appendix F. Ciphers Performance vs. Number of Sub-Images

Table A2. SBTM Results.

SBTM							
Bernard-hermant-nHRXNv2qeDE-unsplash.jpg				Peppers.tiff			
cic1		cic3		cic1		cic3	
h,v sub	avg time ms	h,v sub	avg time ms	h,v sub	avg time ms	h,v sub	avg time ms
[1, 1]	2553.4	[1, 1]	786	[1, 1]	14.1	[1, 1]	6.46
[1, 2]	1009	[1, 2]	434	[1, 2]	5.5	[1, 2]	4.08
[1, 3]	688.3	[1, 3]	397.1	[1, 3]	4.6	[1, 3]	3.44
[1, 4]	543.1	[1, 4]	344.7	[1, 4]	3.2	[1, 4]	3.14
[1, 5]	444.4	[1, 5]	316	[1, 5]	2.5	[1, 5]	3.46
[1, 6]	379.1	[1, 6]	292.7	[1, 6]	3.1	[1, 6]	3.12
[1, 7]	418.1	[1, 7]	286.6	[1, 7]	4.7	[1, 7]	3.22
[1, 8]	397.5	[1, 8]	286.7	[1, 8]	3.2	[1, 8]	3.48
[1, 9]	366.6	[1, 9]	292.2	[1, 9]	3.1	[1, 9]	3.42
[1, 10]	347.2	[1, 10]	289.3	[1, 10]	3.1	[1, 10]	3.84
[2, 1]	981	[2, 1]	434.2	[2, 1]	4.8	[2, 1]	3.14
[2, 2]	523.4	[2, 2]	351.1	[2, 2]	3.1	[2, 2]	3.14
[2, 3]	378.4	[2, 3]	290.7	[2, 3]	3.1	[2, 3]	2.9
[2, 4]	388.5	[2, 4]	283.6	[2, 4]	3.2	[2, 4]	3.14
[2, 5]	366.2	[2, 5]	288.7	[2, 5]	1.5	[2, 5]	3.76
[2, 6]	324.1	[2, 6]	275.4	[2, 6]	3.2	[2, 6]	3.84
[2, 7]	299.3	[2, 7]	265.8	[2, 7]	3.1	[2, 7]	4.4
[2, 8]	288.6	[2, 8]	271.8	[2, 8]	2	[2, 8]	4.86
[2, 9]	272.2	[2, 9]	266	[2, 9]	3.2	[2, 9]	5
[2, 10]	261.6	[2, 10]	262.6	[2, 10]	3.1	[2, 10]	5.74
[3, 1]	679.2	[3, 1]	391.5	[3, 1]	3.1	[3, 1]	2.9
[3, 2]	379.7	[3, 2]	290.8	[3, 2]	3.2	[3, 2]	2.76
[3, 3]	365.4	[3, 3]	291.1	[3, 3]	4.7	[3, 3]	3.52
[3, 4]	331.7	[3, 4]	274.5	[3, 4]	3.1	[3, 4]	3.76
[3, 5]	309.6	[3, 5]	266.1	[3, 5]	1.6	[3, 5]	4.4
[3, 6]	270.6	[3, 6]	262.8	[3, 6]	4.7	[3, 6]	5.6
[3, 7]	271.3	[3, 7]	261.7	[3, 7]	1.6	[3, 7]	5.76
[3, 8]	274.9	[3, 8]	260.7	[3, 8]	3.1	[3, 8]	6.38
[3, 9]	287.9	[3, 9]	258.1	[3, 9]	3.1	[3, 9]	7.06
[3, 10]	280.2	[3, 10]	262.9	[3, 10]	4.8	[3, 10]	7.94
[4, 1]	524.8	[4, 1]	343.1	[4, 1]	1.5	[4, 1]	2.82
[4, 2]	386.7	[4, 2]	283.3	[4, 2]	3.1	[4, 2]	3.14
[4, 3]	323.5	[4, 3]	273.3	[4, 3]	3.7	[4, 3]	3.84
[4, 4]	288.8	[4, 4]	267.3	[4, 4]	1.6	[4, 4]	5.14
[4, 5]	258.3	[4, 5]	261.3	[4, 5]	3.2	[4, 5]	5.28

Table A2. *Cont.*

SBTM							
Bernard-hermant-nHRXNv2qeDE-unsplash.jpg				**Peppers.tiff**			
cic1		**cic3**		**cic1**		**cic3**	
h,v sub	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**
[4, 6]	274.3	[4, 6]	262.7	[4, 6]	3.1	[4, 6]	6.58
[4, 7]	281.6	[4, 7]	261.2	[4, 7]	3.2	[4, 7]	7.34
[4, 8]	280.5	[4, 8]	256.8	[4, 8]	3.1	[4, 8]	8.22
[4, 9]	271.2	[4, 9]	256.5	[4, 9]	3.1	[4, 9]	9.18
[4, 10]	263	[4, 10]	254.9	[4, 10]	4.7	[4, 10]	10.14
[5, 1]	436.7	[5, 1]	316.3	[5, 1]	1.6	[5, 1]	2.5
[5, 2]	357.1	[5, 2]	286.1	[5, 2]	3.1	[5, 2]	3.76
[5, 3]	295.1	[5, 3]	267.4	[5, 3]	1.6	[5, 3]	4.52
[5, 4]	261.9	[5, 4]	261	[5, 4]	3.2	[5, 4]	5.64
[5, 5]	280	[5, 5]	257.5	[5, 5]	3.1	[5, 5]	6.68
[5, 6]	280.3	[5, 6]	257.2	[5, 6]	3.1	[5, 6]	7.92
[5, 7]	276.9	[5, 7]	257.5	[5, 7]	3.2	[5, 7]	8.88
[5, 8]	263.3	[5, 8]	256.8	[5, 8]	3.1	[5, 8]	9.8
[5, 9]	266.1	[5, 9]	253.2	[5, 9]	3.1	[5, 9]	11.04
[5, 10]	266.3	[5, 10]	252	[5, 10]	5.2	[5, 10]	12.04
[6, 1]	379.8	[6, 1]	289.5	[6, 1]	1.6	[6, 1]	3.14
[6, 2]	318.8	[6, 2]	270	[6, 2]	3.1	[6, 2]	3.86
[6, 3]	274.5	[6, 3]	263.7	[6, 3]	1.6	[6, 3]	5.02
[6, 4]	276.7	[6, 4]	259.6	[6, 4]	3.1	[6, 4]	6.68
[6, 5]	285.8	[6, 5]	256.6	[6, 5]	3.2	[6, 5]	7.52
[6, 6]	273.2	[6, 6]	251.9	[6, 6]	3.1	[6, 6]	9.2
[6, 7]	264	[6, 7]	250.2	[6, 7]	3.2	[6, 7]	10.42
[6, 8]	268.2	[6, 8]	250.3	[6, 8]	4.6	[6, 8]	11.78
[6, 9]	267.7	[6, 9]	250	[6, 9]	4.8	[6, 9]	12.92
[6, 10]	275.6	[6, 10]	252.6	[6, 10]	4.7	[6, 10]	14.52
[7, 1]	404.3	[7, 1]	283.4	[7, 1]	1.5	[7, 1]	2.92
[7, 2]	302.5	[7, 2]	262.9	[7, 2]	3.1	[7, 2]	4.38
[7, 3]	273.3	[7, 3]	258.1	[7, 3]	1.6	[7, 3]	5.76
[7, 4]	289.5	[7, 4]	271.1	[7, 4]	3.2	[7, 4]	7.22
[7, 5]	276.7	[7, 5]	266.2	[7, 5]	3.1	[7, 5]	8.86
[7, 6]	263.2	[7, 6]	250.4	[7, 6]	4.2	[7, 6]	10.42
[7, 7]	267.9	[7, 7]	251.8	[7, 7]	4.7	[7, 7]	12
[7, 8]	266.2	[7, 8]	250.2	[7, 8]	4	[7, 8]	13.32
[7, 9]	263.2	[7, 9]	248.9	[7, 9]	3.8	[7, 9]	15.12
[7, 10]	263.3	[7, 10]	249.8	[7, 10]	4.7	[7, 10]	16.78
[8, 1]	384.6	[8, 1]	279.6	[8, 1]	3.2	[8, 1]	3.12

Table A2. *Cont.*

SBTM							
Bernard-hermant-nHRXNv2qeDE-unsplash.jpg				**Peppers.tiff**			
cic1		**cic3**		**cic1**		**cic3**	
h,v sub	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**
[8, 2]	287.9	[8, 2]	266.4	[8, 2]	1.5	[8, 2]	4.48
[8, 3]	278.5	[8, 3]	258.2	[8, 3]	3.1	[8, 3]	6.58
[8, 4]	284.4	[8, 4]	253.4	[8, 4]	3.2	[8, 4]	8.24
[8, 5]	264.6	[8, 5]	249.5	[8, 5]	3.1	[8, 5]	9.82
[8, 6]	267.1	[8, 6]	247.5	[8, 6]	3.2	[8, 6]	11.68
[8, 7]	265	[8, 7]	249.4	[8, 7]	4.7	[8, 7]	13.58
[8, 8]	261.2	[8, 8]	247.6	[8, 8]	4.7	[8, 8]	15.22
[8, 9]	261.4	[8, 9]	250	[8, 9]	4.9	[8, 9]	17.12
[8, 10]	257.3	[8, 10]	252.6	[8, 10]	4.9	[8, 10]	18.98
[9, 1]	364.6	[9, 1]	280.5	[9, 1]	3.1	[9, 1]	3.46
[9, 2]	273.7	[9, 2]	257.6	[9, 2]	2	[9, 2]	5.04
[9, 3]	285.4	[9, 3]	253.8	[9, 3]	2.7	[9, 3]	6.94
[9, 4]	270.3	[9, 4]	253.4	[9, 4]	4.7	[9, 4]	9.26
[9, 5]	263.4	[9, 5]	249	[9, 5]	3.2	[9, 5]	11.06
[9, 6]	264.7	[9, 6]	247.1	[9, 6]	4.7	[9, 6]	12.94
[9, 7]	261.5	[9, 7]	247	[9, 7]	3.1	[9, 7]	14.9
[9, 8]	260	[9, 8]	250.2	[9, 8]	6.3	[9, 8]	17.1
[9, 9]	256.8	[9, 9]	246.7	[9, 9]	6.3	[9, 9]	19.2
[9, 10]	263.1	[9, 10]	248.5	[9, 10]	4.7	[9, 10]	21.16
[10, 1]	343.8	[10, 1]	277.5	[10, 1]	3.1	[10, 1]	3.22
[10, 2]	268.2	[10, 2]	261.5	[10, 2]	1.6	[10, 2]	5.66
[10, 3]	285.2	[10, 3]	258.4	[10, 3]	3.5	[10, 3]	7.62
[10, 4]	273.6	[10, 4]	257.6	[10, 4]	3.1	[10, 4]	10.04
[10, 5]	275.4	[10, 5]	247.4	[10, 5]	4.8	[10, 5]	12.02
[10, 6]	269.9	[10, 6]	248.8	[10, 6]	3.1	[10, 6]	14.6
[10, 7]	269.8	[10, 7]	246.9	[10, 7]	6.3	[10, 7]	16.58
[10, 8]	259.7	[10, 8]	246.7	[10, 8]	4.7	[10, 8]	18.82
[10, 9]	264	[10, 9]	247	[10, 9]	7.8	[10, 9]	21.24
[10, 10]	256.1	[10, 10]	248.6	[10, 10]	6.4	[10, 10]	23.38
min time is	256.1	min time is	246.7	min time is	1.5	min time is	2.5

Table A3. SBTMi Results.

SBTMi							
Bernard-hermant-nHRXNv2qeDE-unsplash.jpg				**Peppers.tiff**			
cic1		**cic3**		**cic1**		**cic3**	
h,v sub	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**
[1, 1]	2504.9	[1, 1]	791.5	[1, 1]	9.9	[1, 1]	6.28
[1, 2]	995.9	[1, 2]	431.4	[1, 2]	5.04	[1, 2]	4.16
[1, 3]	674.5	[1, 3]	400.5	[1, 3]	3.5	[1, 3]	3.46
[1, 4]	519	[1, 4]	352.5	[1, 4]	3.14	[1, 4]	3.14
[1, 5]	433.5	[1, 5]	329	[1, 5]	3.14	[1, 5]	3.22
[1, 6]	383.6	[1, 6]	307.9	[1, 6]	2.52	[1, 6]	3.14
[1, 7]	408.7	[1, 7]	285.1	[1, 7]	2.96	[1, 7]	3.14
[1, 8]	393	[1, 8]	296.4	[1, 8]	2.44	[1, 8]	3.54
[1, 9]	362.6	[1, 9]	294	[1, 9]	2.82	[1, 9]	3.44
[1, 10]	345.7	[1, 10]	292	[1, 10]	2.54	[1, 10]	3.76
[2, 1]	980.6	[2, 1]	428.8	[2, 1]	4.16	[2, 1]	3.2
[2, 2]	519.2	[2, 2]	346.9	[2, 2]	2.84	[2, 2]	2.82
[2, 3]	375.5	[2, 3]	292.7	[2, 3]	2.2	[2, 3]	2.84
[2, 4]	390.5	[2, 4]	291.4	[2, 4]	2.6	[2, 4]	3.22
[2, 5]	343.2	[2, 5]	287.3	[2, 5]	2.5	[2, 5]	3.76
[2, 6]	320.4	[2, 6]	279.3	[2, 6]	2.52	[2, 6]	4.06
[2, 7]	299.4	[2, 7]	271.5	[2, 7]	2.2	[2, 7]	4.5
[2, 8]	286.7	[2, 8]	280.6	[2, 8]	2.28	[2, 8]	4.7
[2, 9]	274.9	[2, 9]	270.1	[2, 9]	2.52	[2, 9]	5.12
[2, 10]	259.8	[2, 10]	267.4	[2, 10]	2.52	[2, 10]	5.96
[3, 1]	671.2	[3, 1]	394.7	[3, 1]	3.12	[3, 1]	2.88
[3, 2]	382.5	[3, 2]	295.7	[3, 2]	2.32	[3, 2]	2.84
[3, 3]	362.6	[3, 3]	291.7	[3, 3]	2.52	[3, 3]	3.44
[3, 4]	320.8	[3, 4]	278	[3, 4]	2.2	[3, 4]	3.84
[3, 5]	295.5	[3, 5]	272.8	[3, 5]	2.54	[3, 5]	4.72
[3, 6]	271.7	[3, 6]	268.8	[3, 6]	2.26	[3, 6]	5.42
[3, 7]	269.5	[3, 7]	262.2	[3, 7]	2.52	[3, 7]	5.64
[3, 8]	274.9	[3, 8]	263.9	[3, 8]	2.82	[3, 8]	6.6
[3, 9]	279.8	[3, 9]	263.6	[3, 9]	2.52	[3, 9]	7.28
[3, 10]	284.7	[3, 10]	261	[3, 10]	2.94	[3, 10]	7.84
[4, 1]	517	[4, 1]	347	[4, 1]	2.82	[4, 1]	2.58
[4, 2]	395.3	[4, 2]	287.5	[4, 2]	2.18	[4, 2]	3.46
[4, 3]	321.6	[4, 3]	276.2	[4, 3]	2.52	[4, 3]	4.08
[4, 4]	288.7	[4, 4]	273.1	[4, 4]	2.4	[4, 4]	4.8
[4, 5]	260.6	[4, 5]	265.1	[4, 5]	2.4	[4, 5]	5.64
[4, 6]	275.4	[4, 6]	266	[4, 6]	2.82	[4, 6]	6.36
[4, 7]	282.5	[4, 7]	262.8	[4, 7]	2.82	[4, 7]	7.62

Table A3. *Cont.*

SBTMi							
Bernard-hermant-nHRXNv2qeDE-unsplash.jpg				Peppers.tiff			
cic1		cic3		cic1		cic3	
h,v sub	avg time ms	h,v sub	avg time ms	h,v sub	avg time ms	h,v sub	avg time ms
[4, 8]	278.1	[4, 8]	258.3	[4, 8]	2.94	[4, 8]	8.24
[4, 9]	269.3	[4, 9]	261.2	[4, 9]	3.14	[4, 9]	9.12
[4, 10]	263.5	[4, 10]	257.5	[4, 10]	3.24	[4, 10]	10.1
[5, 1]	432.2	[5, 1]	316.3	[5, 1]	2.5	[5, 1]	2.6
[5, 2]	342.1	[5, 2]	284.5	[5, 2]	2.2	[5, 2]	3.76
[5, 3]	297.2	[5, 3]	270	[5, 3]	2.52	[5, 3]	4.54
[5, 4]	258.9	[5, 4]	263.7	[5, 4]	2.5	[5, 4]	5.64
[5, 5]	279.5	[5, 5]	261.3	[5, 5]	2.66	[5, 5]	6.74
[5, 6]	279.9	[5, 6]	260.5	[5, 6]	2.82	[5, 6]	7.92
[5, 7]	269.8	[5, 7]	258.5	[5, 7]	2.82	[5, 7]	8.8
[5, 8]	262.7	[5, 8]	259	[5, 8]	3.46	[5, 8]	10.12
[5, 9]	261.6	[5, 9]	255.8	[5, 9]	3.54	[5, 9]	11.14
[5, 10]	266.8	[5, 10]	255.4	[5, 10]	4.08	[5, 10]	12.32
[6, 1]	370.4	[6, 1]	290.9	[6, 1]	2.2	[6, 1]	2.82
[6, 2]	318.7	[6, 2]	276.6	[6, 2]	2.3	[6, 2]	3.76
[6, 3]	272.5	[6, 3]	269	[6, 3]	2.52	[6, 3]	5.42
[6, 4]	275.1	[6, 4]	262.4	[6, 4]	2.56	[6, 4]	6.34
[6, 5]	284.2	[6, 5]	259.3	[6, 5]	2.76	[6, 5]	7.86
[6, 6]	267.7	[6, 6]	255.2	[6, 6]	3.24	[6, 6]	9.18
[6, 7]	261.4	[6, 7]	262.1	[6, 7]	3.44	[6, 7]	10.42
[6, 8]	268.6	[6, 8]	259.5	[6, 8]	3.76	[6, 8]	11.68
[6, 9]	263.1	[6, 9]	261.3	[6, 9]	4.04	[6, 9]	13.04
[6, 10]	266.2	[6, 10]	262.1	[6, 10]	4.4	[6, 10]	14.2
[7, 1]	403.5	[7, 1]	288.8	[7, 1]	2.38	[7, 1]	3.14
[7, 2]	299.9	[7, 2]	270.9	[7, 2]	2.1	[7, 2]	4.22
[7, 3]	272.3	[7, 3]	266.9	[7, 3]	2.52	[7, 3]	5.9
[7, 4]	280.5	[7, 4]	270.5	[7, 4]	2.82	[7, 4]	7.3
[7, 5]	270.2	[7, 5]	262.5	[7, 5]	3.2	[7, 5]	8.86
[7, 6]	262.3	[7, 6]	259	[7, 6]	3.46	[7, 6]	10.12
[7, 7]	269.7	[7, 7]	252.6	[7, 7]	3.76	[7, 7]	11.92
[7, 8]	266.9	[7, 8]	255.8	[7, 8]	4.16	[7, 8]	13.58
[7, 9]	259.2	[7, 9]	253.2	[7, 9]	4.78	[7, 9]	15.14
[7, 10]	261.4	[7, 10]	253.4	[7, 10]	4.74	[7, 10]	16.48
[8, 1]	379.5	[8, 1]	283.1	[8, 1]	2.2	[8, 1]	3.14
[8, 2]	285.4	[8, 2]	268.4	[8, 2]	2.52	[8, 2]	5
[8, 3]	279.4	[8, 3]	261.5	[8, 3]	2.52	[8, 3]	6.36

Table A3. *Cont.*

SBTMi							
Bernard-hermant-nHRXNv2qeDE-unsplash.jpg				**Peppers.tiff**			
cic1		**cic3**		**cic1**		**cic3**	
h,v sub	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**	**h,v sub**	**avg time ms**
[8, 4]	280.9	[8, 4]	259.1	[8, 4]	2.72	[8, 4]	8.24
[8, 5]	266.1	[8, 5]	257.9	[8, 5]	3.46	[8, 5]	9.84
[8, 6]	267.5	[8, 6]	256.9	[8, 6]	3.78	[8, 6]	11.68
[8, 7]	269.3	[8, 7]	258.2	[8, 7]	4.18	[8, 7]	13.62
[8, 8]	266.5	[8, 8]	259.2	[8, 8]	4.4	[8, 8]	15.24
[8, 9]	266.1	[8, 9]	257.3	[8, 9]	5.12	[8, 9]	17.12
[8, 10]	260.1	[8, 10]	265.7	[8, 10]	5.68	[8, 10]	18.6
[9, 1]	363.3	[9, 1]	286.5	[9, 1]	2.18	[9, 1]	3.22
[9, 2]	270.8	[9, 2]	272.7	[9, 2]	2.6	[9, 2]	5.32
[9, 3]	281.4	[9, 3]	262.1	[9, 3]	2.52	[9, 3]	7
[9, 4]	267.1	[9, 4]	260.6	[9, 4]	3.14	[9, 4]	8.86
[9, 5]	260.4	[9, 5]	259.2	[9, 5]	3.54	[9, 5]	11.06
[9, 6]	265.7	[9, 6]	258.6	[9, 6]	4.4	[9, 6]	12.92
[9, 7]	260.5	[9, 7]	257	[9, 7]	4.5	[9, 7]	14.9
[9, 8]	258.8	[9, 8]	264.3	[9, 8]	5.34	[9, 8]	17.04
[9, 9]	256.3	[9, 9]	258.2	[9, 9]	5.78	[9, 9]	18.98
[9, 10]	260.6	[9, 10]	258	[9, 10]	6.3	[9, 10]	21.18
[10, 1]	342.8	[10, 1]	282.6	[10, 1]	2.5	[10, 1]	3.22
[10, 2]	259.4	[10, 2]	263.9	[10, 2]	2.3	[10, 2]	5.64
[10, 3]	279.5	[10, 3]	257.2	[10, 3]	2.92	[10, 3]	7.62
[10, 4]	261	[10, 4]	252.6	[10, 4]	3.36	[10, 4]	9.8
[10, 5]	271.6	[10, 5]	249.5	[10, 5]	3.54	[10, 5]	12
[10, 6]	264.4	[10, 6]	250.7	[10, 6]	4.72	[10, 6]	14.26
[10, 7]	268.7	[10, 7]	251.6	[10, 7]	4.82	[10, 7]	16.64
[10, 8]	261.2	[10, 8]	252	[10, 8]	6.02	[10, 8]	18.84
[10, 9]	268.3	[10, 9]	255.7	[10, 9]	6.06	[10, 9]	21.02
[10, 10]	259.2	[10, 10]	253.2	[10, 10]	6.98	[10, 10]	23.14
min time is	256.3	min time is	249.5	min time is	2.1	min time is	2.58

References

1. Zia, U.; McCartney, M.; Scotney, B.; Martinez, J.; AbuTair, M.; Memon, J.; Sajjad, A. Survey on image encryption techniques using chaotic maps in spatial, transform and spatiotemporal domains. *Int. J. Inf. Secur.* **2022**, *21*, 917–935. [CrossRef]
2. Shah, P.; Ayoade, J. An Empricial Study of Brute Force Attack on Wordpress Website. In Proceedings of the 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 23–25 January 2023; pp. 659–662.
3. Hurley, N.; Cheng, Z.; Zhang, M. Statistical attack detection. In Proceedings of the Third ACM Conference on Recommender Systems, New York, NY, USA, 23–25 October 2009; pp. 149–156.
4. Pal, D.; Ali, M.; Das, A.; Roy Chowdhury, D. A cluster-based practical key recovery attack on reduced-round AES using impossible-differential cryptanalysis. *J. Supercomput.* **2023**, *79*, 6252–6289. [CrossRef]
5. Attaullah Shah, T.; Jamal, S. An improved chaotic cryptosystem for image encryption and digital watermarking. *Wirel. Pers. Commun.* **2020**, *110*, 1429–1442. [CrossRef]

6. Zhang, X.; Hu, Y. Multiple-image encryption algorithm based on the 3D scrambling model and dynamic DNA coding. *Opt. Laser Technol.* **2021**, *141*, 107073. [CrossRef]
7. Zhang, X.; Hu, Y. An overview of digital audio steganography. *IETE Tech. Rev.* **2020**, *37*, 632–650.
8. Bhowmik, S.; Acharyya, S. Image cryptography: The genetic algorithm approach. In Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering, Shanghai, China, 10–12 June 2011; Volume 2, pp. 223–227.
9. Wang, X.; Su, Y.; Liu, C.; Li, J.; Li, S.; Cai, Z.; Wan, W. Security enhancement of image encryption method based on Fresnel diffraction with chaotic phase. *Opt. Commun.* **2022**, *506*, 127544. [CrossRef]
10. Chen, Y.; Xie, S.; Zhang, J. A hybrid domain image encryption algorithm based on improved henon map. *Entropy* **2022**, *24*, 287. [CrossRef]
11. Zhou, W.; Wang, X.; Wang, M.; Li, D. A new combination chaotic system and its application in a new Bit-level image encryption scheme. *Opt. Lasers Eng.* **2022**, *149*, 106782. [CrossRef]
12. Lai, Q.; Zhang, H.; Kuate, P.; Xu, G.; Zhao, X. Analysis and implementation of no-equilibrium chaotic system with application in image encryption. *Appl. Intell.* **2022**, *52*, 11448–11471. [CrossRef]
13. Matthews, R. On the derivation of a "chaotic" encryption algorithm. *Cryptologia* **1989**, *13*, 29–42. [CrossRef]
14. Bin Faheem, Z.; Ali, A.; Khan, M.A.; Ul-Haq, M.; Ahmad, W. Highly dispersive substitution box (S-box) design using chaos. *ETRI J.* **2020**, *42*, 619–632. [CrossRef]
15. Niu, Y.; Zhou, Z.; Zhang, X. An image encryption approach based on chaotic maps and genetic operations. *Multimed. Tools Appl.* **2020**, *79*, 25613–25633. [CrossRef]
16. Hosny, K.M.; Kamal, S.T.; Darwish, M.M. Novel encryption for color images using fractional-order hyperchaotic system. *J. Ambient Intell. Humaniz. Comput.* **2022**, *13*, 973–988. [CrossRef] [PubMed]
17. Patel, S.; Thanikaiselvan, V.; Pelusi, D.; Nagaraj, B.; Arunkumar, R.; Amirtharajan, R. Colour image encryption based on customized neural network and DNA encoding. *Neural Comput. Appl.* **2021**, *33*, 14533–14550. [CrossRef]
18. Zhou, Y.; Bao, L.; Chen, C.P. A new 1D chaotic system for image encryption. *Signal Process.* **2014**, *97*, 1172–1182. [CrossRef]
19. Chen, J.; Horng, S. Novel SCAN-CA-based image security system using SCAN and 2-D von Neumann cellular automata. *Signal Process. Image Commun.* **2010**, *25*, 413–426. [CrossRef]
20. Maniccam, S.; Bourbakis, G. Lossless image compression and encryption using SCAN. *Pattern Recognit.* **2001**, *34*, 1229–1245. [CrossRef]
21. Shyu, S. Image encryption by multiple random grids. *Pattern Recognit.* **2009**, *42*, 1582–1596. [CrossRef]
22. Chen, T.; Li, K. Multi-image encryption by circular random grids. *Inf. Sci.* **2012**, *189*, 55–265. [CrossRef]
23. Li, L.; Abd El-Latif, A.; Niu, X. Elliptic curve ElGamal based homomorphic image encryption scheme for sharing secret images. *Signal Process.* **2012**, *92*, 1069–1078. [CrossRef]
24. Zhou, Y.; Panetta, K.; Agaian, S.; Chen, C.P. (n, k, p)-Gray code for image systems. *IEEE Trans. Cybern.* **2013**, *43*, 515–529. [CrossRef]
25. Liao, X.; Lai, S.; Zhou, Q. A novel image encryption algorithm based on self-adaptive wave transmission. *Signal Process.* **2010**, *90*, 2714–2722. [CrossRef]
26. Chen, T.; Wu, C. Compression-unimpaired batch-image encryption combining vector quantization and index compression. *Inf. Sci.* **2010**, *180*, 1690–1701. [CrossRef]
27. Bhatnagar, G.; Wu, Q.; Raman, B. A new fractional random wavelet transform for fingerprint security. *IEEE Trans. Syst. Man Cybern. Part A Syst. Humans* **2011**, *42*, 262–275. [CrossRef]
28. ur Rehman, A.; Liao, X.; Ashraf, R.; Abdullah, S.; Wang, H. A color image encryption technique using exclusive-OR with DNA complementary rules based on chaos theory and SHA-2. *Optik* **2018**, *159*, 348–367. [CrossRef]
29. Seyedzadeh, M.; Mirzakuchaki, S. A fast color image encryption algorithm based on coupled two-dimensional piecewise chaotic map. *Signal Process.* **2012**, *92*, 1202–1215. [CrossRef]
30. Tong, J.; Zhang, M.; Wang, Z.; Liu, Y.; Xu, H.; Ma, J. A fast encryption algorithm of color image based on four-dimensional chaotic system. *J. Vis. Commun. Image Represent.* **2015**, *33*, 219–234. [CrossRef]
31. Li, Y.; Wang, C.; Chen, H. A hyper-chaos-based image encryption algorithm using pixel-level permutation and bit-level permutation. *Opt. Lasers Eng.* **2017**, *90*, 238–246. [CrossRef]
32. Abd El-Latif, A.; Li, L.; Wang, N.; Han, Q.; Niu, X. A new approach to chaotic image encryption based on quantum chaotic system, exploiting color spaces. *Signal Process.* **2013**, *93*, 2986–3000. [CrossRef]
33. Liu, H.; Kadir, A.; Niu, Y. Chaos-based color image block encryption scheme using S-box. *AEU-Int. J. Electron. Commun.* **2014**, *68*, 676–686. [CrossRef]
34. Mazloom, S.; Eftekhari-Moghadam, A. Color image encryption based on coupled nonlinear chaotic map. *Chaos Solitons Fractals* **2009**, *42*, 1745–1754. [CrossRef]
35. Kadir, A.; Aili, M.; Sattar, M. Color image encryption scheme using coupled hyper chaotic system with multiple impulse injections. *Optik* **2017**, *129*, 231–238. [CrossRef]
36. Al-Daraiseh, A.; Sanjalawe, Y.; Al-E'mari, S.; Fraihat, S.; Bany Taha, M.; Al-Muhammed, M. Cryptographic Grade Chaotic Random Number Generator Based on Tent-Map. *J. Sens. Actuator Netw.* **2023**, *12*, 73. [CrossRef]
37. Daoui, A.; Yamni, M.; Chelloug, S.; Wani, M.; El-Latif, A. Efficient image encryption scheme using novel 1D multiparametric dynamical tent map and parallel computing. *Mathematics* **2023**, *7*, 1589. [CrossRef]

38. Choi, J.; Seok, S.; Seo, H.; Kim, H. A fast ARX model-based image encryption scheme. *Multimed. Tools Appl.* **2016**, *75*, 14685–14706. [CrossRef]
39. Li, C.; Luo, G.; Li, C. A parallel image encryption algorithm based on chaotic Duffing oscillators. *Multimed. Tools Appl.* **2018**, *77*, 19193–19208. [CrossRef]
40. Yuan, H.; Liu, Y.; Lin, T.; Hu, T.; Gong, L.H. A new parallel image cryptosystem based on 5D hyper-chaotic system. *Signal Process. Image Commun.* **2017**, *52*, 87–96. [CrossRef]
41. You, L.; Yang, E.; Wang, G. A novel parallel image encryption algorithm based on hybrid chaotic maps with OpenCL implementation. *Soft Comput.* **2020**, *24*, 12413–12427. [CrossRef]
42. He, Y.; Zhang, Y.; He, X.; Wang, X. A new image encryption algorithm based on the OF-LSTMS and chaotic sequences. *Sci. Rep.* **2021**, *11*, 6398. [CrossRef]
43. Wang, X.; Su, Y. Color image encryption based on chaotic compressed sensing and two-dimensional fractional Fourier transform. *Sci. Rep.* **2020**, *10*, 18556. [CrossRef]
44. Lee, W.; Phan, R.; Yap, W.; Goi, B. SPRING: A novel parallel chaos-based image encryption scheme. *Nonlinear Dyn.* **2018**, *92*, 575–593. [CrossRef]
45. Luo, Y.; Zhou, R.; Liu, J.; Cao, Y.; Ding, X. A parallel image encryption algorithm based on the piecewise linear chaotic map and hyper-chaotic map. *Nonlinear Dyn.* **2018**, *93*, 1165–1181. [CrossRef]
46. Wang, X.; Feng, L.; Zhao, H. Fast image encryption algorithm based on parallel computing system. *Inf. Sci.* **2019**, *486*, 340–358. [CrossRef]
47. Wang, X.; Zhao, H. Fast image encryption algorithm based on parallel permutation-and-diffusion strategy. *Multimed. Tools Appl.* **2020**, *79*, 19005–19024. [CrossRef]
48. Nkandeu, Y.; Mboupda Pone, R.; Tiedeu, A. Image encryption algorithm based on synchronized parallel diffusion and new combinations of 1D discrete maps. *Sens. Imaging* **2020**, *21*, 55. [CrossRef]
49. Zhu, S.; Deng, X.; Zhang, W.; Zhu, C. Image encryption scheme based on newly designed chaotic map and parallel DNA coding. *Mathematics* **2023**, *11*, 231. [CrossRef]
50. Mozaffari, S. Parallel image encryption with bitplane decomposition and genetic algorithm. *Multimed. Tools Appl.* **2018**, *77*, 25799–25819. [CrossRef]
51. Zhang, Y. Statistical test criteria for sensitivity indexes of image cryptosystems. *Inf. Sci.* **2021**, *550*, 313–328. [CrossRef]
52. Chaudhary, N.; Shahi, T.; Neupane, A. Secure image encryption using chaotic, hybrid chaotic and block cipher approach. *J. Imaging* **2022**, *8*, 167. [CrossRef]