Journal of Information Hiding and Multimedia Signal Processing ©2025 ISSN 2073-4212
Ubiquitous International Volume **16**, Number **2**, June **2025**

A New Asymmetric Image Encryption Algorithm Based on Integer Chebyshev Map and Henon-Sine Map

Rinaldi[1,*]

[1]School of Electrical Engineering and Informatics,
Institut Teknologi Bandung
rinaldi@staff.stei.itb.ac.id

[*]Corresponding author: Rinaldi

Abstract. *Chaos systems have become a promising alternative for developing secure and robust image encryption algorithms. However, most chaos-based encryption methods are symmetric, meaning encryption and decryption use the identical key. This makes the sender and receiver have to share the secret key before doing communication, making it vulnerable to interception by attackers. To address this issue, asymmetric algorithms can be used. This paper introduced a new asymmetric image encryption algorithm that integrates the integer Chebyshev map, the Henon-Sine map, and an enhanced RC4 stream cipher. The integer Chebyshev map, defined on $\mathbb{Z}_p$, p is a prime number, is used for public and private key generation of the users, while the Henon-Sine map is used to produce the secret key, for image encryption and decryption, derived from the improved RC4 stream cipher. Experiments have been conducted and the results show that the proposed algorithm is secure against brute-force attacks, correlation attacks, statistical attacks, entropy attacks, and sensitivity attacks, while also maintaining robustness against common image processing techniques.*
Keywords: Image encryption, chaos, Chebyshev Map, Henon-Sine map, asymmetric

1. **Introduction.** In this age of information technology and the internet, messages in the form of text, images, audio, and video are represented digitally. These messages can be easily transmitted through communication channels or stored in various storage media. However, confidential messages must be protected from unauthorized access and manipulation. Encryption is a technique used to transform the messages into a form that cannot be interpreted by unauthorized parties.

Image encryption is designed to protect digital images from being interpreted by unauthorized individuals. Encrypted images cannot be understood or perceived unless a valid key is used for decryption. Image encryption can be performed using traditional cryptographic algorithms—either symmetric or asymmetric—such as AES, RC4, RSA, ECC, ElGamal, and others, or by employing specialized encryption algorithms for images. Recent advancements in image encryption involve utilizing chaotic systems, which provide a high level of randomness essential for robust encryption. The chaotic systems have an important characteristic: they are sensitive to initial parameter values. This sensitivity means that even a slight alteration in initial values (such as the starting point or system parameters) can produce a vastly different chaotic sequence, which results in a strong diffusion effect in the encryption algorithm [1]. These chaotic systems exhibit random behavior and are extremely difficult to predict, analyze, or control.

Several chaotic maps have been applied in cryptographic algorithms, such as the Logistic Map, Sine Map, Henon Map, Baker Map, Chebyshev Map, Lorenz 96 Map [2] and others. In image encryption algorithms, these maps are typically used to generate sequences of random numbers. These random numbers serve various purposes, such as generating encryption keys, permuting pixels, and determining encryption parameters. Much work has been done by researchers to design encryption algorithms for images based on chaotic systems. Comprehensive reviews of image encryption algorithms using chaotic systems can be found in [3, 4].

In cryptography, based on the type of key used, encryption algorithms are grouped into two classes: symmetric and asymmetric algorithms. In symmetric algorithms, both encryption and decryption use the identical key. Before sending and receiving messages, the sender and receiver must share this key to encrypt and decrypt the message. In contrast, asymmetric algorithms use different keys for encryption and decryption. The sender encrypts the message using the receiver's public key, while the receiver decrypts it using his (or her) private key. There is no need to share a secret key between the sender and receiver, as both have their own private and public key pairs. Asymmetric algorithms are also referred to as public-key cryptography.

Therefore, image encryption algorithms that use chaotic systems are also grouped into two classes: symmetric and asymmetric image encryption algorithms. However, most research focuses on symmetric-key image encryption algorithms, with only a few papers proposing asymmetric ones. Some of these are summarized as follows. Obbaid & Al Saffar designed an asymmetric image encryption method by applying a cubic curve combined with a standard map [5]. The method is considered asymmetric because it employs the Elliptic Curve Diffie-Hellman (ECDH) algorithm, a public-key cryptography technique, to produce the initial values of the map. Shakiba proposed an asymmetric algorithm using the Chebyshev polynomial map and a one-time pad to encrypt color images, with the Chebyshev map generating the same secret key for both the sender and receiver [6]. Ye et al. introduced an asymmetric encryption method by applying a 3-D logistic map. The algorithm used the RSA algorithm (an asymmetric cryptography technique) to generate the initial keys for the 3-D logistic map [7]. Liu & Ye in [8] introduced an asymmetric image encryption method based on a combination of the logistic map and sine map to generate a secret key, with the RSA algorithm encrypting the secret key. Additional asymmetric algorithms can be found in Geetha & Kumar [9] and Simon & Varghese [10].

The weakness of the proposed image encryption methods mentioned above is that they do not truly perform encryption and decryption in the sense of public key cryptography, where the image is encrypted using the public key and decrypted using the private key. Instead, image encryption and decryption still rely on a shared secret key for the sender and receiver. In these cases, the sender and receiver generate a shared secret key using an asymmetric cryptography algorithm (RSA, ElGamal, ECDH, etc.) combined with a chaotic system. The image encryption and decryption are then performed using the shared secret key.

Therefore, using a public key cryptography algorithm to encrypt the secret key does not make the image encryption algorithm truly asymmetric. In this paper, we introduce an asymmetric image encryption method based on the Chebyshev Map and the Henon-Sine Map. The algorithm is considered asymmetric because it uses the public key and the private key to do encryption and decryption. In this method, the sender and receiver only need to input their public and private keys during the encryption/decryption process. The method will then generate a same key for encrypting and decrypting the image. The Chebyshev Map is used to generate the public and private keys, as it is commonly applied in asymmetric algorithms [11, 12]. The Henon-Sine map is used in secret key generation

by adopting keystream generation in the Improved RC4 (IRC4) stream cipher [13]. The Improved RC4 stream cipher is a modified version of RC4 to address the weakness of the original RC4 which was vulnerable to correlation attacks. Therefore, the proposed asymmetric image encryption method can overcome the problem of sharing secret keys inherent in symmetric encryption methods.

This paper is organized into five sections. The first section is the introduction. The second section describes the related work. The third section presents the proposed method. The fourth section details the experiment and discussion of the results, and the fifth section is the conclusion.

2. Related Work.

The proposed method uses two chaotic maps, namely the Chebyshev Map and the Henon-Sine Map. These maps form the basis for the asymmetric image encryption algorithm. This section describes both chaotic maps.

2.1. Henon-Sine Map.

The Henon-Sine map combines two chaotic systems: the Henon map and the Sine map. This combination was designed to overcome the limitations of the individual maps [14]. Despite their simplicity, both the Henon map and Sine map share a significant weakness: their trajectories are relatively easy to predict. To address this issue, Wu et al. [14] developed a hyperchaotic system known as the 2D Henon-Sine map, described as follows:

$$\begin{cases} x_{n+1} = 1 - a\,(\sin x_n)^2 + y_n \bmod 1 \\ y_{n+1} = b x_n \bmod 1 \end{cases} \tag{1}$$

where a and b are parameters of the map. The 2D Henon-Sine map shows chaos behavior when $a \leq 0.71$ or $a \geq 0.71$ and $b = 0.7$. The trajectory of the 2D Henon-Sine map is evenly distributed across the entire plane (see Figure 1).

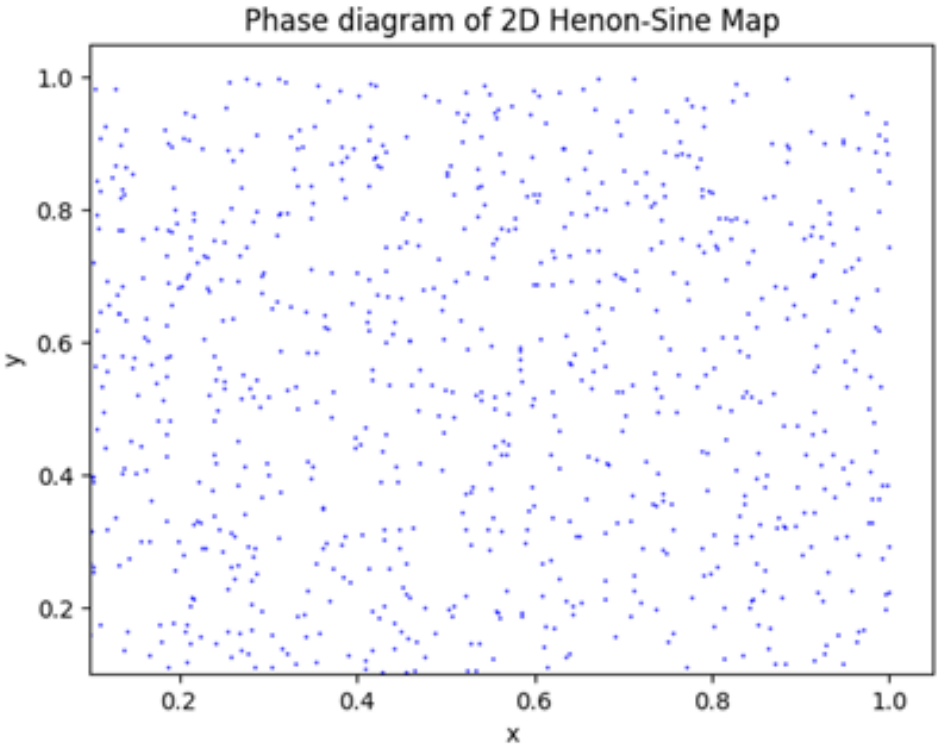


FIGURE 1. Phase diagram of 2D Henon-Sine Map

The 2D Henon-Sine map can also be written in the form one-dimensional Henon-Sine map as

$$x_{n+1} = 1 - a\,(\sin x_n)^2 + bx_{n-1} \bmod 1 \tag{2}$$

This map needs two initial values (x_0 and x_1) which can be considered as the secret keys.

2.2. **Chebyshev Map.** The Chebyshev polynomial is a chaotic map known for exhibiting random-like behavior under certain initial conditions. Mathematically, the Chebyshev polynomial of degree n is defined recursively as follows:

$$C_{n+1}(x) = \begin{cases} 2xC_n(x) - C_{n-1}(x), & n > 1 \\ x, & n = 1 \\ 1, & n = 0 \end{cases} \tag{3}$$

The Chebyshev polynomial maps $C_n : R \to R$. The most important property of the Chebyshev polynomial is the composition of

$$C_r(C_s(x)) = C_{rs}(x) \tag{4}$$

for arbitrary degree r and s. The composition is commutative, namely

$$C_r(C_s(x)) = C_s(C_r(x)) = C_{rs}(x) \tag{5}$$

If we restrict x in interval $[-1, 1]$ only, then $C_n(x) \in [-1, 1]$, thus the interval $[-1, 1]$ is invariant under mapping $C_n : [-1, 1] \to [-1, 1]$. The Chebyshev polynomial is defined in the interval $[-1, 1]$, and for all $n > 1$ the Chebyshev polynomial is a chaotic map with positive Lyapunov exponent of $\lambda = \ln n$. For $n = 2$, Chebyshev polynomial becomes to a logistic map, a well known chaotic map [12].

Based on property (4), Kocarev et al. designed an asymmetric encryption method as follows [11]. Suppose Alice and Bob are communicating by sending messages. Alice selects a large integer s as her private key. Next, Alice generates a random number $x \in [-1, 1]$ and then she computes $C_s(x)$. Her public key is $(x, C_s(x))$, which she sends to Bob. On Bob's side, Bob selects a large integer r as his private key. He computes his public key $(x, C_r(x))$ and sends it to Alice. Bob encrypts a message to Alice as follows: first, Bob converts the message as number $P \in [-1, 1]$. Next, he computes $C_{rs} = C_r(C_s(x))$, encrypt the message as $Ciphertext = P * C_{rs}$, and sends it to Alice. Alice decrypts the message as follows: at first Alice computes $C_{sr} = C_s(C_r(x))$, then Alice recovers the plaintext by computing $M = Ciphertext/C_{rs}$.

According to the algorithm above, C_{rs} is the shared secret key used Alice and Bob to encrypt and decrypt message symmetrically. The shared secret key is common to all asymmetric encryption algorithms based on the chaotic system. Therefore, the process of computing the shared secret key is similar to the Diffie-Hellman key exchange algorithm. Both Alice and Bob each generate their private keys, s and r, respectively. Alice computes $C_s(x)$, send $(x, C_s(x))$ to Bob. Bob computes $C_r(x)$ and send it to Alice. Now, Alice and Bob can compute the symmetric key: Alice computes $C_{sr} = C_s(C_r(x))$, Bob computes $C_{rs} = C_r(C_s(x))$, and $C_{sr} = C_{rs}$. They use the symmetric key, C_{rs}, to encrypt plaintext M by

$$Ciphertext = M * C_{rs} \tag{6}$$

and reversely decrypt ciphertext $Ciphertext$ by

$$M = Ciphertext/C_{rs}. \tag{7}$$

3. **Proposed Method.** Unfortunately, the asymmetric encryption algorithm based on Chebyshev Map has a problem of computational precision. The algorithm operates on real numbers, whereas computers represent real numbers in a finite number of digits, depending on precision level. This representation introduces round-off error when the real number cannot be specified within the finite number of digits. The precision problem of real numbers can lead to security problem [15].

Since cryptography operates on integer arithmetics, we define the Chebyshev Map on $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ where p is a prime number. Therefore, we modify Chebyshev map into an integer Chebyshev map as follows:

$$C_{n+1}(x) = \begin{cases} 2xC_n(x) - C_{n-1}(x) \bmod p, & n > 1 \\ x \bmod p, & n = 1 \\ 1, & n = 0 \end{cases} \tag{8}$$

where $x \in \mathbb{Z}_p$. It can easily be verified that

$$C_r(C_s(x) \bmod p) = C_{rs}(x) \bmod p \tag{9}$$

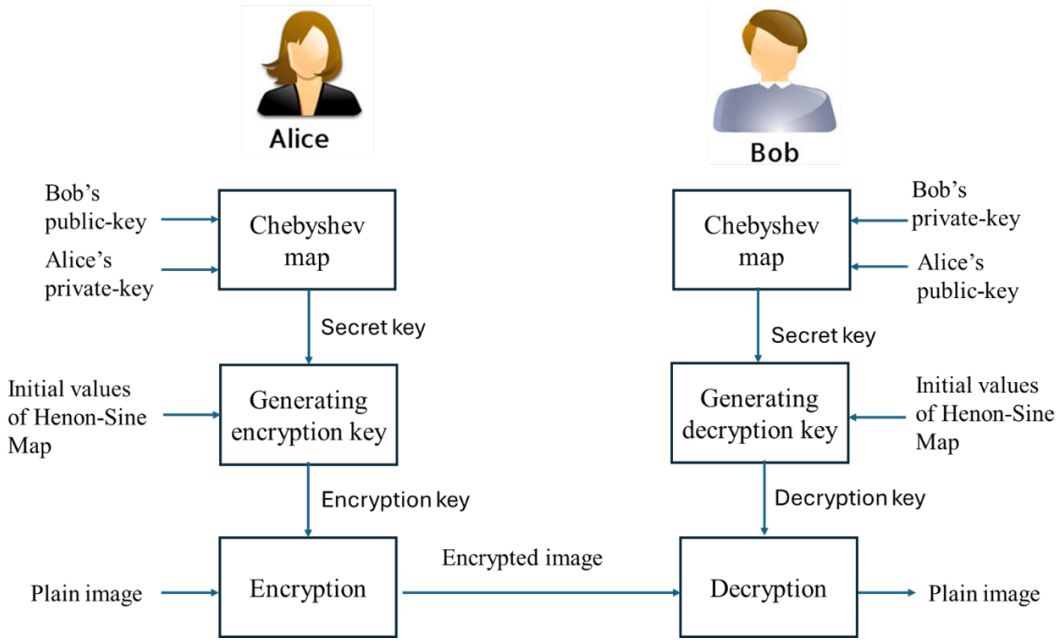


FIGURE 2. Process diagram for image encryption and decryption

We now propose an asymmetric image encryption method based on the Chebyshev map, Henon-Sine map, and Improved RC4. Figure 2 illustrates the process diagram for image encryption and decryption using the proposed scheme. Suppose Alice sends the secret image (plain image) to Bob. Alice encrypts the image using Bob's public key and her public key, sends the encrypted image to Bob, and Bob decrypts the encrypted image using his private key and Alice's public key. The scheme has three processes: key generation (public key and private key), image encryption, and image decryption. Each process is described in each sub-section below.

3.1. **Generate Public Key and Private Key.** The original Improved RC4 stream cipher will be used to generates the encryption key. It accepts input of the secret key with a maximum length of 256 characters. If the key length is less than 256 characters, the characters in the key will be repeated. Assuming the key length is n ($n \leq 256$), both Alice and Bob generate n public keys and n private keys using the Chebyshev map. Initially

Alice and Bob agree on n, a large prime number p, and sequence x of n elements. Both n, p, and x are all public and can communicated over a public channel. Alice and Bob generate their own public keys and private keys using Chebyshev map as in Algorithm 1.

Algorithm 1 Generates the public key and the private key of n elements

Input: length of key (n), prime number (p), and sequence of integer $x[1..n]$
Output: private key ($r[1..n]$) and public key ($Cr[1..n]$)

input($r[1..n]$) ▷ Choose n integers as private keys
for $i \leftarrow 0$ to n **do** ▷ Initialize Chebyshev polynomial, $C0(x) = 1$, $C1(x) = x \bmod p$
  $C \leftarrow 1$
  $C1 \leftarrow x[i] \bmod p$ ▷ Compute Chebyshev polynomial $C(x)$ using recursive formula
  for $k \leftarrow 2$ to $r[i]$ **do**
    $Ck \leftarrow (2 * x[i] * C1 - C) \bmod p$ ▷ $C_k(x) = 2 * x * C_{k-1}(x) - C_{k-2}(x) \bmod p$
    $C \leftarrow C1$
    $C1 \leftarrow Ck$
  end for
  $Cr[i] \leftarrow Ck$
end for

3.2. **Image encryption.** Image encryption is carried out using the receiver's public key. However, to generate an encryption key, the sender's private key is also required. Therefore, Alice encrypts the image using Bob's public key and her own private key. The details of the image encryption methods are shown in Algorithm 2. In the algorithm, Alice's private key is denoted as r, and Bob's public key as Cs. The algorithm generates the secret key ($SK[1..n]$) using Eq. (9). Once the secret key is generated, the encryption key (U) is produced by adopting the keystream generation in the Improved RC4 (IRC4). Here we apply the Henon-Sine map in the table randomization process for keystream generation. The Henon-Sine map uses two initial values taken from the average value of half of the elements in the secret key normalized by the total of all elements.:

$$x1 = mean(SK[1..n/2])/sum(SK[1..n]) \tag{10}$$

$$x2 = mean(SK[n/2 + 1..n])/sum(SK[1..n]) \tag{11}$$

Based on the Improved RC4, we scramble the table *State* as follows:

for $i \leftarrow 0$ to 255 **do** ▷ Scramble the table State
  $x \leftarrow HenonSineMap(a, b, x1, x2)$
  $x2 \leftarrow x1$
  $x1 \leftarrow x$
  $n1 \leftarrow Encode(x)$
  $x \leftarrow HenonSineMap(a, b, x1, x2)$
  $x2 \leftarrow x1$
  $x1 \leftarrow x$
  $n2 \leftarrow Encode(x)$
  $j \leftarrow (n1 + State[n2] + SK[n2 \bmod n]) \bmod 256$
  $swap(State[i], State[j])$
end for

Since the *HenonSine* map produces random floating-point numbers between 0 and 1, the *Encode* function transforms these real numbers into integers between 0 and 255. Next,

iterate $M \times N$ times to generate the encryption key U. Finally, the encryption key U is XORed with the plain image F, to yield the encrypted image:

$$O = F \oplus U \tag{12}$$

Pseudo-code of image encryption is shown on Algorithm 2 below.

Algorithm 2: Encrypts the image

Input: plain image (F), prime number (p), length of key (n), receiver's public key (Cs), sender's private key (r)
Output: encrypted image (O)

1: // Generate a secret key, SK
2: **for** $i \leftarrow 1$ **to** n **do**
3: $x[i] \leftarrow Ts[i]$ // Initialize Chebyshev polynomial, $C_0(x) = 1$, $C_1(x) = x \bmod p$
4: $C \leftarrow 1$
5: $C1 \leftarrow x[i] \bmod p$
6: // Compute Chebyshev polynomial $C(x)$ using recursive formula
7: **for** $k \leftarrow 2$ **to** $r[i]$ **do**
8: $Ck \leftarrow (2 \times x[i] \times C1 - C) \bmod p$
9: $C \leftarrow C1$
10: $C1 \leftarrow Ck$
11: **end for**
12: $SK[i] \leftarrow Ck \bmod 256$
13: **end for**
14: // Generate the encryption key, U, by using the secret key, SK
15: **for** $k \leftarrow 0$ **to** 255 **do**
16: $State[k] \leftarrow k$
17: **end for**
18: $j \leftarrow 0$
19: $x1 \leftarrow \text{mean}(SK[1 : \lfloor n/2 \rfloor])/\text{mean}(SK)$
20: $x2 \leftarrow \text{mean}(SK[\lfloor n/2 \rfloor + 1 : n])/\text{mean}(SK)$
21: **for** $k \leftarrow 0$ **to** 255 **do**
22: $x \leftarrow HenonSineMap(a, b, x1, x2)$
23: $x2 \leftarrow x1$
24: $x1 \leftarrow x$
25: $n1 \leftarrow Encode(x)$
26: $x \leftarrow HenonSineMap(a, b, x1, x2)$
27: $x2 \leftarrow x1$
28: $x1 \leftarrow x$
29: $n2 \leftarrow Encode(x)$
30: $j \leftarrow (n1 + State[n2] + SK[n2 \bmod n]) \bmod 256$
31: $swap(State[k], State[j])$
32: **end for**
33: $j \leftarrow 0$
34: $k \leftarrow 0$
35: $[v, w] \leftarrow size(F)$ // size of image F is $v \times w$
36: **for** $row \leftarrow 1$ **to** v **do**
37: **for** $col \leftarrow 1$ **to** w **do**

38: // Produce the encryption key, U
39: $j \leftarrow (j+1) \bmod 256$
40: $k \leftarrow (k + State[j]) \bmod 256$
41: $swap(State[j], State[k])$
42: $U[row, col] \leftarrow State[(State[j] + State[k]) \bmod 256]$
43: **end for**
44: **end for**
45: $O \leftarrow F \oplus U$

3.3. **Image decryption.** Unlike encryption, image decryption is carried out using the receiver's private key. However, to produce a symmetric key, the sender's public key is also required. In this scenario, Bob decrypts the image using his private key and Alice's public key. The details of the image decryption method are shown in Algorithm 3. In the algorithm, Bob's private key is s, and Alice's public key is Cr. Similar to the encryption process, the decryption algorithm first generates the secret key SK using Eq. (9). Once the secret key is generated, the decryption key (U) is produced by adopting the keystream generation in the Improved RC4. Finally, we perform XOR operation between the decryption key, U, and the encrypted image, O, to yield the decrypted image:

$$F = O \oplus U \tag{13}$$

Pseudo-code of image decryption is shown on Algorithm 3 below.

Algorithm 3: Decrypts the image

Input: encrypted image (O), prime number (p), length of key (n), receiver's private key (s), sender's public key (Cr)
Output: original image (F)

1: // Generate a secret key, SK
2: **for** $i \leftarrow 1$ **to** n **do**
3: $x[i] \leftarrow Cr[i]$ // Initialize Chebyshev polynomial, $C_0(x) = 1$, $C_1(x) = x \bmod p$
4: $C \leftarrow 1$
5: $C1 \leftarrow x[i] \bmod p$
6: // Compute Chebyshev polynomial $C(x)$ using recursive formula
7: **for** $k \leftarrow 2$ **to** $s[i]$ **do**
8: $Ck \leftarrow (2 \times x[i] \times C1 - C) \bmod p$
9: $C \leftarrow C1$
10: $C1 \leftarrow Ck$
11: **end for**
12: $SK[i] \leftarrow Ck \bmod 256$
13: **end for**
14: // Generate the decryption key, U, by using the secret key, SK
15: **for** $k \leftarrow 0$ **to** 255 **do**
16: $State[k] \leftarrow k$
17: **end for**
18: $j \leftarrow 0$
19: $x1 \leftarrow \text{mean}(SK[1 : \lfloor n/2 \rfloor])/\text{mean}(SK)$
20: $x2 \leftarrow \text{mean}(SK[\lfloor n/2 \rfloor + 1 : n])/\text{mean}(SK)$
21: **for** $k \leftarrow 0$ **to** 255 **do**

```
        x ← HenonSineMap(a, b, x1, x2)
        x2 ← x1
        x1 ← x
        n1 ← Encode(x)
        x ← HenonSineMap(a, b, x1, x2)
        x2 ← x1
        x1 ← x
        n2 ← Encode(x)
        j ← (n1 + State[n2] + SK[n2 mod n]) mod 256
        swap(State[k], State[j])
    end for
    j ← 0
    k ← 0
    [v, w] ← size(F)   // size of image F is v × w
    for row ← 1 to v do
        for col ← 1 to w do
            // Produce the decryption key, U
            j ← (j + 1) mod 256
            k ← (k + State[j]) mod 256
            swap(State[j], State[k])
            U[row, col] ← State[(State[j] + State[k]) mod 256]
        end for
    end for
    F ← O ⊕ U
```

3.4. **Security of the algorithm.** The public key algorithm proposed above has the following public parameters: a prime number (p), the key length (n), a sequence of integers (x), the receiver's public key sequence (Ts), and the sender's public key sequence (Tr). To decrypt the encrypted image, an attacker would need to know the receiver's private key sequence (s). Ts and Tr represent Chebyshev polynomials of orders s and r, respectively. While computing (x, Ts) using Eq. (4) is straightforward, computing s given x and $Ts(x)$ is much more difficult. To recover s, the attacker would need to compute $T_u(x)$ for all $u \geq 2$ until they find a u such that $T_u(x) = Ts(x)$. This attack becomes infeasible if s is a very large number. For example, if s is 2048 bits in length, the attacker would need to compute $T_u(x)$ for all u from $2, 3, \ldots, 2^{2048} - 1$. Since s is an integer sequence of length n, the attacker would have to repeat this process n times. Hence, the algorithm is resistant to brute force attacks.

4. **Experiments and Results.** We have implemented the proposed algorithm as a program in MATLAB platform. We tested the program on twelve standard images, both grayscale and color images (RGB images), as shown in Figure 3. All images are 512 × 512 in size.

The proposed method can be applied for encryption and decryption of both grayscale and color images. For grayscale images, encryption and decryption are performed as in Algorithm 1 and Algorithm 2 above. For color images in the RGB color space, encryption is performed each color channel separately. Each channel uses the same keystream (U), although it is possible to use different keystreams for each channel. Therefore, for color images, the encryption and decryption parts in Algorithm 2 (line 42) and Algorithm 3 (line 42) are modified as follows:

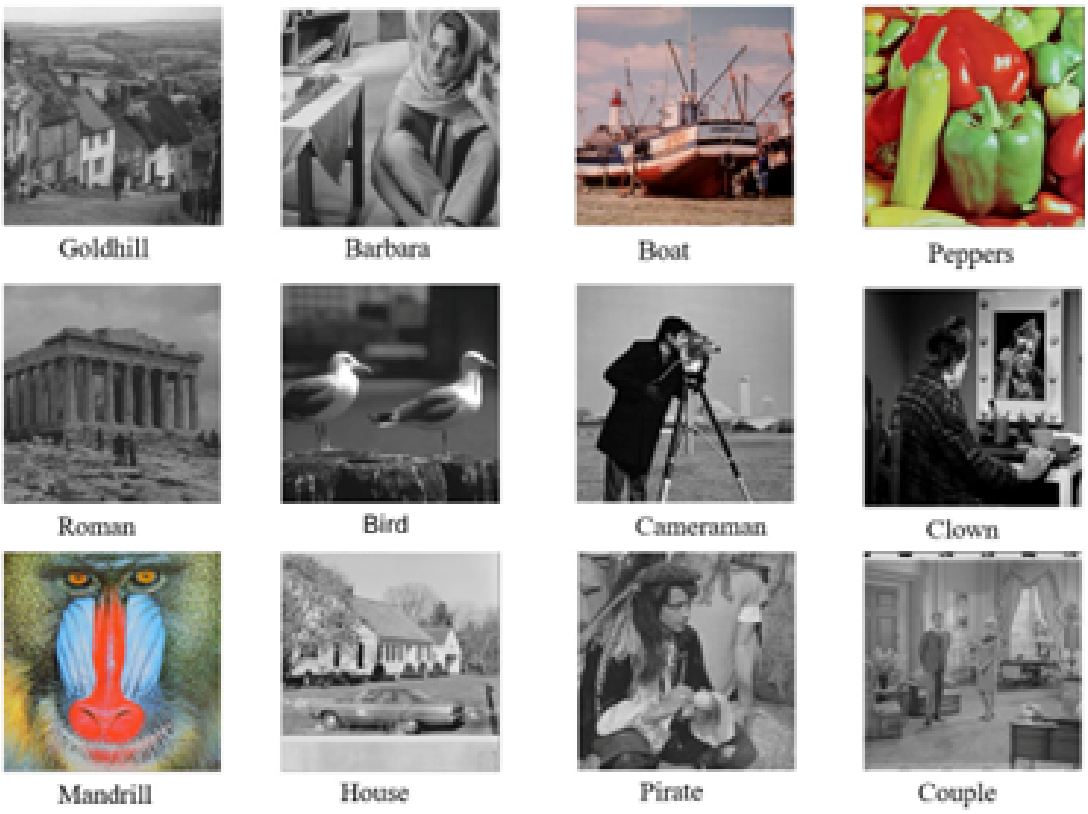


FIGURE 3. Twelve images used for testing

Encryption stage:

42: for channel ← 1 to 3 do
43: for row ← 1 to v do
44: for col ← 1 to w do
45: $O[row, col, channel] \leftarrow F[row, col, channel] \oplus U[row, col]$
46: end for
47: end for
48: end for

Decryption stage:

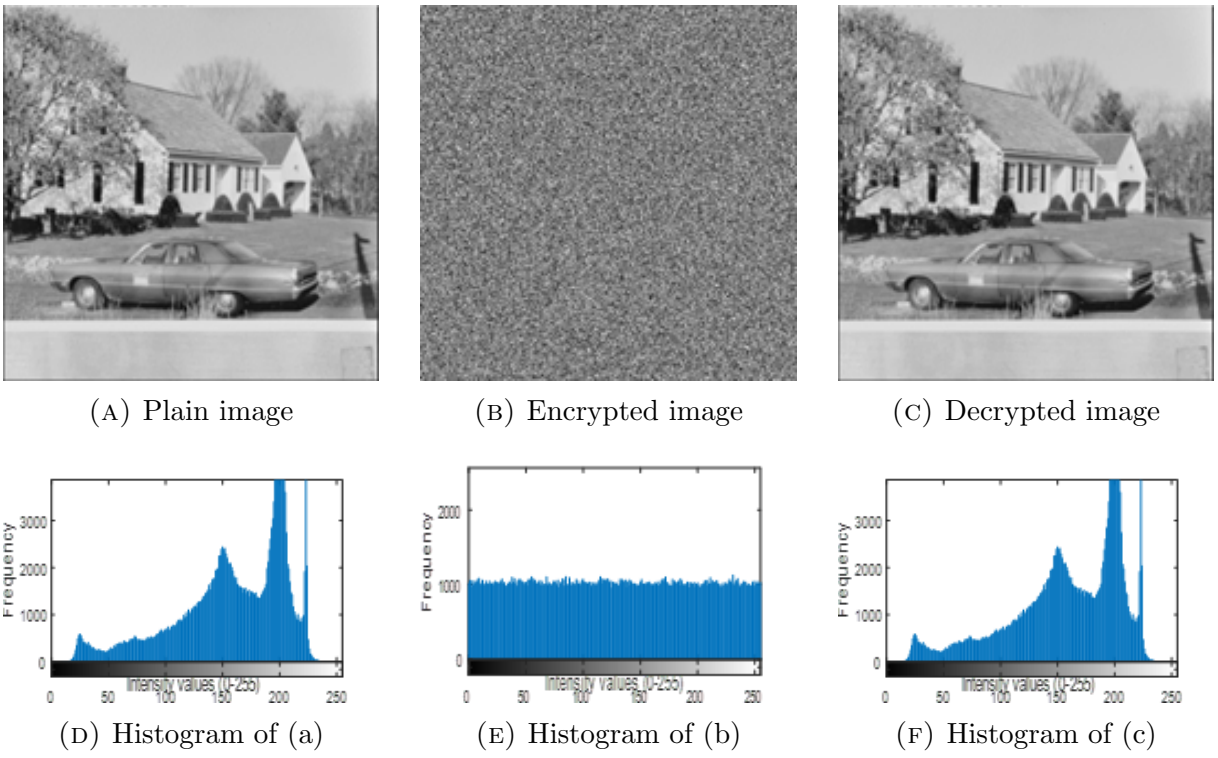
42: for channel ← 1 to 3 do
43: for row ← 1 to v do
44: for col ← 1 to w do
45: $F[row, col, channel] \leftarrow O[row, col, channel] \oplus U[row, col]$
46: end for
47: end for
48: end for

We conducted the experiments for all test image, and the results are explained in each sub-section below.

4.1. **Generating public key and private key.** Think Alice and Bob want to communicate by sending a secret image to each other using the asymmetric version of the RC4 cipher described above. First, Alice and Bob must each have their public and private keys. In these experiments, to produce the public and private keys, suppose Alice and Bob agree on $n = 8$, $p = 1051$, and $x = [1456, 7658, 2875, 9123, 5908, 1012, 2873, 8742]$. Alice chooses

her private key as $s = [1540, 1430, 1234, 8765, 9081, 1542, 7432, 9235]$. Using Eq. (1), Alice computes her public key and obtains $Ts = [71, 711, 487, 820, 730, 58, 433, 859]$. On the other hand, Bob chooses his private key as $r = [1370, 7865, 9082, 1276, 2387, 8752, 8542, 9009]$. Bob computes his public key using Eq. (1) and obtains $Tr = [474, 690, 905, 235, 920, 739, 555, 519]$. We use these private and public keys in the next experiments.

4.2. **Histogram analysis.** An image encryption algorithm must be secure against statistical attacks. The distribution of pixels in the encrypted image should make it difficult for an attacker to use frequency analysis methods to deduce the key. Therefore, the pixels in the encrypted image should have a uniform distribution. The histogram of the encrypted image should appear flat. Figure 4 shows the results for the grayscale image, include plain image, the encrypted image, the decrypted image, and the histograms of the corresponding images.



(A) Plain image (B) Encrypted image (C) Decrypted image

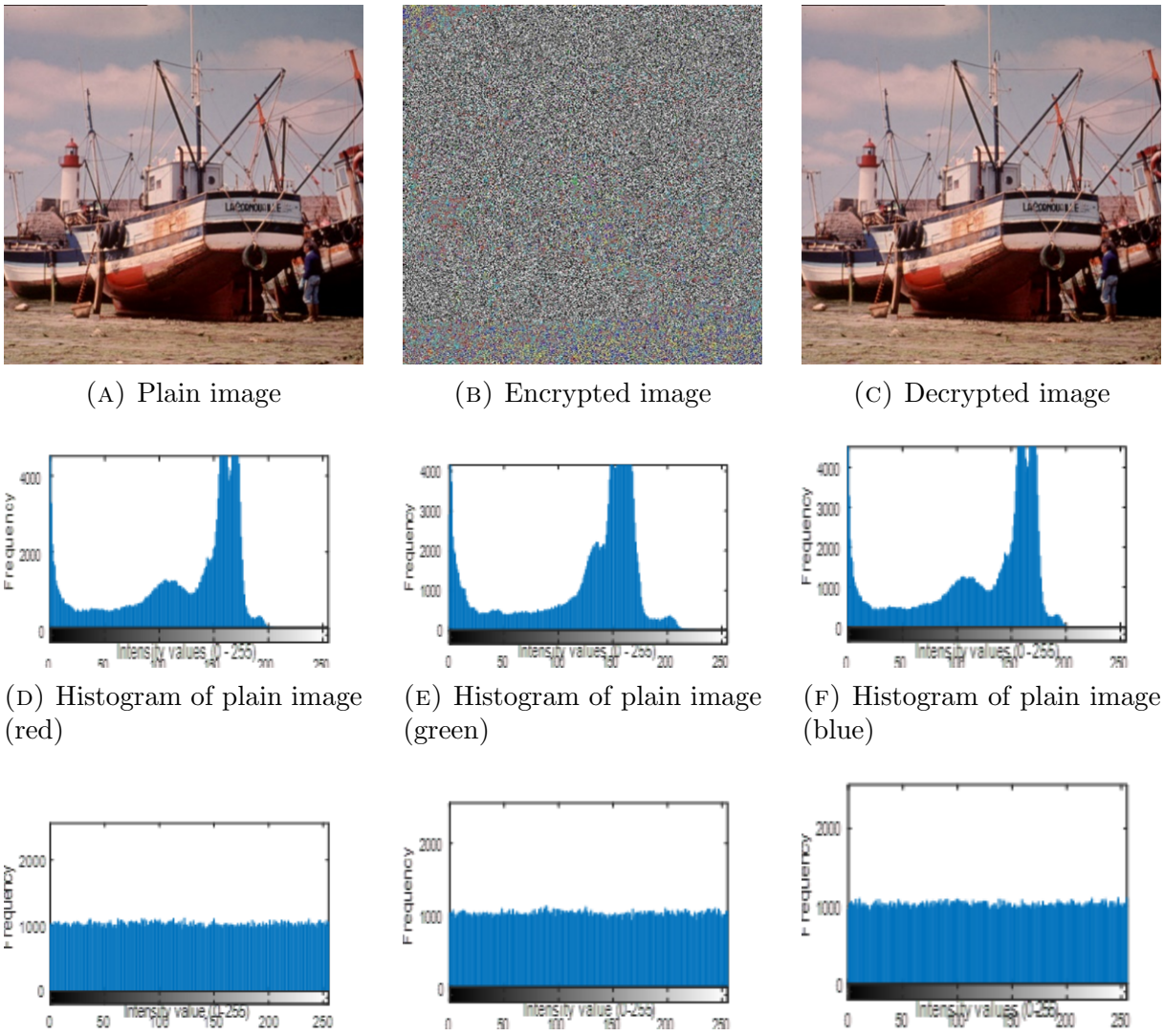
(D) Histogram of (a) (E) Histogram of (b) (F) Histogram of (c)

FIGURE 4. Histogram of the plain image and the encrypted image for a grayscale image

Next, Figure 5 shows the results for for the color image. Since a color image has three color channels (red-green-blue), a histogram is created for each channel. The decrypted image is the same as the original plain image.

From Figure 4 and Figure 5 we can observe visually that the histograms of the encrypted images (Figure 4(e) and Figure 5(g-i)) appear almost flat, indicating that the proposed method is secure against statistical attacks. It makes it difficult for cryptanalysts to deduce the secret keys or pixel values of the plain images.

4.3. **Sensitivity analysis.** An important characteristic of a chaotic map is its sensitivity to even the smallest changes in the initial values. A single bit change in the initial condition results in a significant change in the system. The proposed method exhibits this sensitivity. If we change one bit of the key, decryption of the image failed to transform it back to the original image. Figure 6 shows the effect of a slight change in the receiver's



(A) Plain image (B) Encrypted image (C) Decrypted image

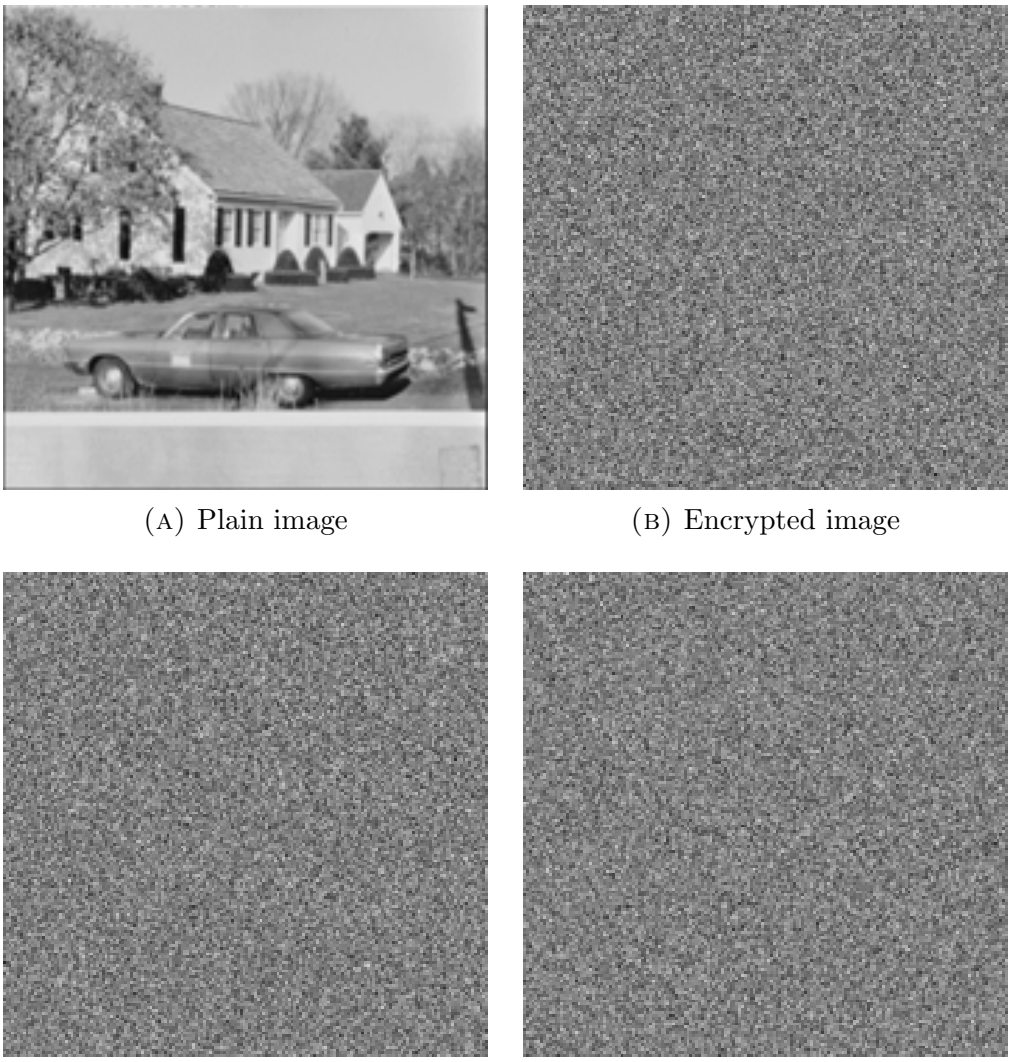
(D) Histogram of plain image (red) (E) Histogram of plain image (green) (F) Histogram of plain image (blue)

(G) Histogram of encrypted image (red) (H) Histogram of encrypted image (green) (I) Histogram of encrypted image (blue)

FIGURE 5. Histogram of the plain image and the encrypted image for a color image

private key. We change the least significant bit of an element in the private key $s = [1540, 1430, 1234, 8765, 9081, 1542, 7432, 9235]$, incrementing it (9235 becomes 9236) and decrementing it (9235 becomes 9234). The decrypted images, shown in Figure 6, still appear as random images.

4.4. **Correlation analysis.** Correlation in an image describes the linear relationship between pixels. Neighboring pixels in a plain image have a strong correlation, while pixels in an encrypted image have a weak correlation. Figure 7 shows the correlation distribution of adjacent pixels in the plain image and the encrypted image of the 'house' image. The correlation of pixels in the plain image is clustered around the diagonal line, indicating a strong correlation, whereas the correlation of pixels in the encrypted image is evenly distributed across the plane, indicating that the correlation between pixels has been eliminated. The correlation between pixels in an image is measured by the correlation coefficient, defined as



(A) Plain image (B) Encrypted image

(C) Decrypted image ($s = s - 1$) (D) Decrypted image ($s = s + 1$)

FIGURE 6. Sensitivity effect of the proposed method for a sample image

$$r_{XY} = \frac{\text{cov}(X, Y)}{\sqrt{D(X)}\sqrt{D(Y)}} \tag{14}$$

where X and Y are random pixels,

$$\text{cov}(X, Y) = \frac{1}{n}\sum_{i=1}^{n}[x_i - E(X)][y_i - E(Y)] \tag{15}$$

$$D(X) = \frac{1}{n}\sum_{i=1}^{n}[x_i - E(X)]^2 \tag{16}$$

and

$$E(X) = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{17}$$

In our experiment, we randomly selected 1,000 pixels from both the plain images and the encrypted images, and then computed the correlation coefficients between horizontally, vertically, and diagonally adjacent pixels in each image. The correlation coefficients for

each image are shown in Table 1. From the table, the correlation coefficients of pixels in the plain images, on all directions, are close to 1, indicating a strong correlation, whereas the correlation coefficients in the encrypted images, on all directions, are close to zero, indicating a weak correlation. We also comparative study with other methods [4-7] for the common test images. Other methods also give similar results, namely the correlation coefficients of the encrypted images, in all directions, is always close to zero. Comparison of results with other methods can only be done if using the same test images, therefore the empty part in the Table 1 (also Table 2) indicates that comparison of results with other methods cannot be done.

TABLE 1. Correlation coefficients of the plain image and the encrypted image, and comparison with other methods (for some common test images)

Image		Correlation coefficients			Comparison with other methods			
		Hor.	Vert.	Diag.	[4]	[5]	[6]	[7]
Goldhill	P-image	0.9703	0.9599	0.9435				
	E-image	-0.011	0.0305	0.0037				
Roman	P-image	0.9702	0.9649	0.9410				
	E-image	0.053	0.0535	0.0505				
Mandrill	P-image	0.8802	0.7836	0.7388				
	E-image	0.0140	-0.0456	0.0085	-0.0024 0.0053 0.0017			
House	P-image	0.9536	0.9622	0.9169				
	E-image	0.0187	0.0242	-0.0070				
Bird	P-image	0.9704	0.9598	0.9412				
	E-image	0.0463	-0.0145	-0.0201				
Boat	P-image	0.9545	0.9773	0.9465				
	E-image	0.0096	0.0461	-0.0272				0.0079, -0.0034, -0.0037
Couple	P-image	0.9324	0.8625	0.8149				
	E-image	0.0247	0.0220	0.0007				
Barbara	P-image	0.8940	0.9565	0.8845				
	E-image	0.0324	0.0210	0.0103				
Pirate	P-image	0.9732	0.9818	0.9634				
	E-image	0.0221	0.0472	-0.0495			0.0033, 0.0055, -0.0249	-0.0023, -0.0071, -0.0039
Cameraman	P-image	0.9837	0.9974	0.9772				
	E-image	-0.009	0.0248	-0.0113				
Peppers	P-image	0.9764	0.9699	0.9480				
	E-image	0.0251	0.0616	0.0081	0.0026, -0.0004, -0.0027	0.0105, -0.0235, -0.0212	0.0020, -0.0041, 0.0085	-0.0043, -0.0051, 0.0046
Clown	P-image	0.9780	0.9865	0.9681				
	E-image	0.0054	0.0159	-0.0502				

Note: P-image = Plain image, E-image = Encrypted image
Hor = Horizontal; Ver = Vertical, Diag = Diagonal

4.5. **Entropy analysis.** Entropy refers to the disorder in a system. A message M consisting of n symbols m_i (where $i = 0, 1, ..., n$) and the probability of each symbol $p(m_i)$

Plain image *Encrypted image*

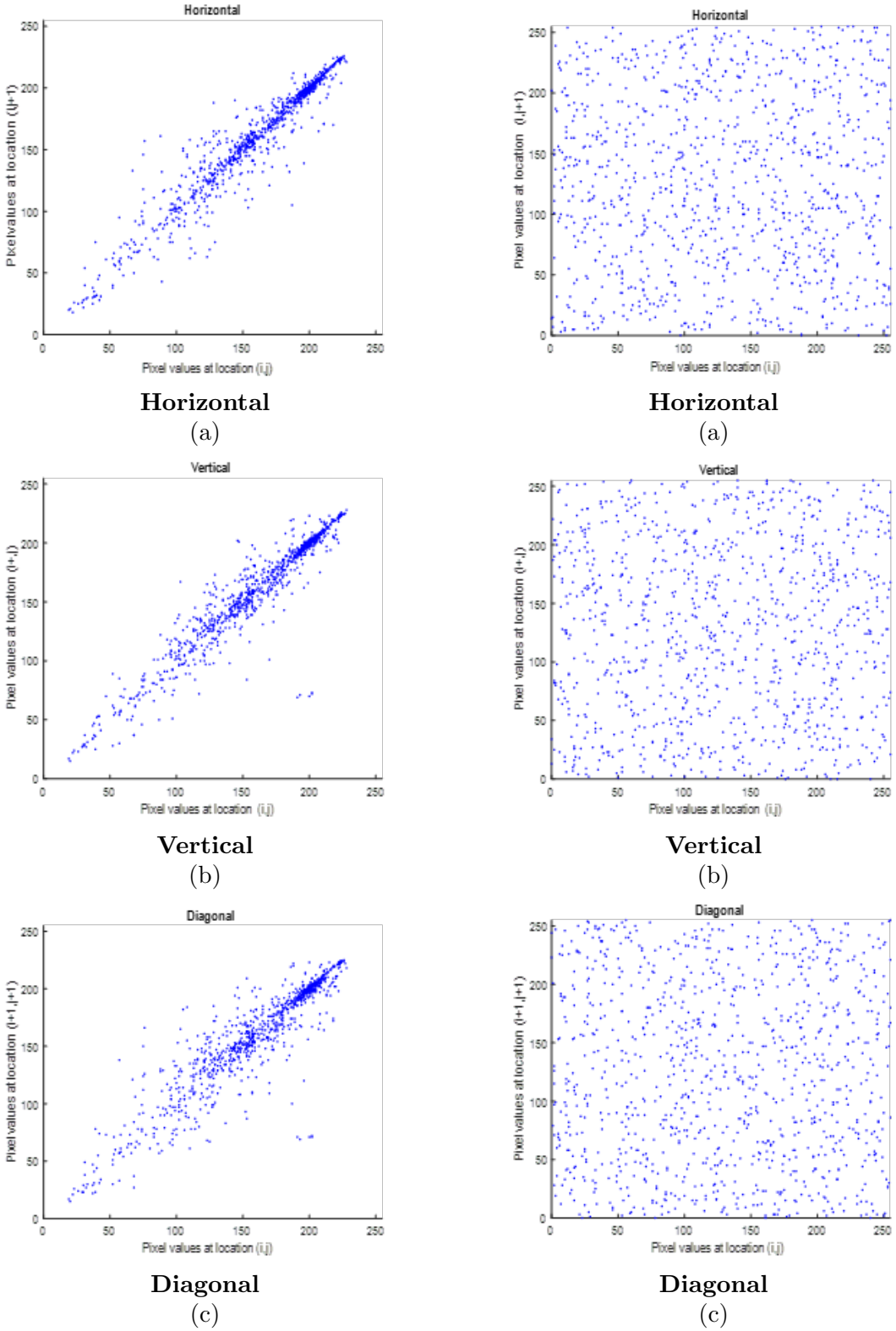


FIGURE 7. Correlation distribution of adjacent pixels on plain image (left) and encrypted image (right)

has an entropy measured by the formula:

$$H(M) = -\sum_{i=0}^{n-1} p(m_i) \log_2 p(m_i) \tag{18}$$

For an image with 256 gray levels, the symbols are $m_0 = 0, m_1 = 1, \ldots, m_{255} = 255$ and $p(m_i)$ can be obtained from histogram. Therefore, the entropy of the image is

$$H(M) = -\sum_{i=0}^{255} p(m_i) \log_2 p(m_i) \tag{19}$$

The higher the entropy value, the greater the disorder in a message, and the higher the degree of unpredictability. Since an encrypted image can be viewed as a random message, its ideal entropy should be equal to eight. A plain image, being non-random, has an entropy less than eight. An entropy value lower than eight indicates a higher security risk because it increases the predictability of the image.

The entropies of the plain images and the encrypted images are summarized in Table 2. Note that the entropy of the encrypted images is close to eight, while the entropy of the plain images is always less than eight. In Table 2 we also compare the results with other methods for some test images. Our method is better than [4] and [5] and only slightly different in results (about 10^{-4}) with [7] and [8]. Overall, the experiment results shows that the proposed method is secure against entropy-based attacks.

TABLE 2. Entropy of the plain images and the encrypted images and comparison with other methods (for encrypted images only)

Image	Entropy		Comparison with other methods			
	Plain-image	Encrypted image	[4]	[5]	[6]	[7]
Goldhill	7.4778	7.9992				
Roman	6.1808	7.9993				
Mandrill	7.3579	7.9992	7.9974	7.9991		7.9994
House	7.2416	7.9993				
Bird	5.8484	7.9991				
Boat	7.1238	7.9991			7.9994	7.9994
Couple	7.2010	7.9992				
Barbara	7.6321	7.9991	7.9975			7.9998
Pirate	7.2367	7.9992			7.9998	7.9999
Cameraman	7.0480	7.9994				
Peppers	7.5712	7.9994	7.9970	7.9992	7.9994	7.9994
Clown	5.3684	7.9991				

4.6. **Robustness analysis.** We tested whether the encrypted images are robust to common image processing techniques, such as image noising and image compression. In real-world applications, such image processing is common. Image noising was performed by adding 1% salt and pepper noise to the encrypted image, followed by decryption. Image compression was tested by compressing the encrypted images into JPEG format and then decompressing them back into bitmap format. Figure 8 shows the encrypted image of Clown's image after noise addition and the encrypted image of Pirate's image after compression. The decrypted images contain noise, but they are still well recognizable.

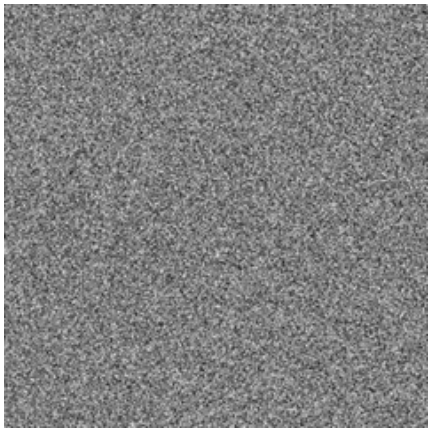
We measured PSNR of the decrypted images using formula

$$PSNR = 20 \times \log_{10} \left(\frac{255}{rms} \right) \tag{20}$$

where

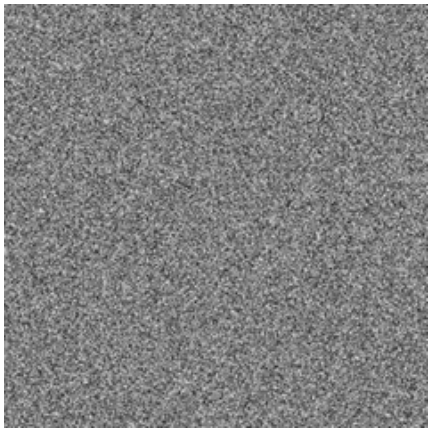
$$rms = \sqrt{\frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}(I_{ij} - \hat{I}_{ij})^2} \tag{21}$$

M and N are image size, I and $\hat{I}$ are the original image and the decrypted image, respectively. PSNR of the decrypted images are summarized in Table 3. All PSNRs are below 30 but the decrypted images can still be recognized well.



(a) Encrypted image ("Clown") with salt & pepper noise

(b) Decrypted of noisy encrypted image



(c) Encrypted image ("Pirate") after JPEG compression

(d) Decrypted of compressed image

FIGURE 8. Robustness of the encrypted image after noise attack and compression attack

TABLE 3. PSNR of the decrypted images after adding noise and after image compression

Image	PSNR	
	Salt & pepper noise	Compression
Goldhill	29.0306	22.4720
Roman	29.6095	22.8939
Mandrill	29.6874	22.7717
House	28.6620	22.4123
Bird	27.9335	22.0539
Boat	29.0515	22.9810
Couple	29.6134	22.2730
Barbara	28.9078	22.4215
Pirate	29.0527	22.7045
Cameraman	28.4679	21.5937
Peppers	28.7004	22.5151
Clown	26.7017	19.9667

5. **Conclusion.** An asymmetric image encryption algorithm based on the Chebyshev map and the Henon-Sine map is proposed in this paper. The Chebyshev map over Z_p, where p is a prime number, is used to generate the public and private keys for the users. The Henon-Sine map and the improved RC4 algorithm are employed to generate the encryption and decryption keys. The proposed algorithm has been tested on a set of sample images. Experimental results demonstrate that the algorithm is secure against brute-force attacks, provided the key parameters are sufficiently large integers. It is also resistant to entropy attacks, correlation attacks, and statistical attacks, while exhibiting high sensitivity to even a one-bit change in the key. Furthermore, the algorithm is robust against noise and compression attacks.

REFERENCES

[1] L. Kocarev, "Chaos-Based Cryptography: A Brief Overview," *IEEE Circuits and Systems Magazine*, vol. 1, no. 3, 2001.

[2] H. T. Elshoush, "A Novel Approach to Improve the Performance of Serpent Algorithm using Lorenz 96 Chaos-based Block Key Generation," *Journal of Information Hiding and Multimedia Signal Processing*, vol. 13, no. 1, Mar. 2022.

[3] U. Zia, M. McCartney, B. Scotney, J. Martinez, M. AbuTair, J. Memon, and A. Sajjad, "Survey on image encryption techniques using chaotic maps in spatial, transform and spatiotemporal domains," *International Journal of Information Security*, vol. 21, pp. 917–935, 2022.

[4] B. Zhang and L. Liu, "Chaos-Based Image Encryption: Review, Application, and Challenges," *Mathematics*, vol. 11, p. 2585, 2023.

[5] M. J. Obaid and N. F. H. Al Saffar, "Asymmetric Image Encryption Based on Singular Cubic Curve with Chaotic Map," *Iraqi Journal of Science*, vol. 65, no. 5, pp. 2605–2618, 2024.

[6] A. Shakiba, "A randomized CPA-secure asymmetric-key chaotic color image encryption scheme based on the Chebyshev mappings and one-time pad," *Journal of King Saud University – Computer and Information Sciences*, vol. 33, pp. 562–571, 2021.

[7] G. Ye, H. S. Wu, X. L. Huang, and S. Y. Tan, "Asymmetric image encryption algorithm based on a new three-dimensional improved logistic chaotic map," *Chinese Physics B*, vol. 32, p. 030504, 2023.

[8] S. Liu and G. Ye, "Asymmetric image encryption algorithm using a new chaotic map and an improved radial diffusion," *Optik – International Journal for Light and Electron Optics*, vol. 28, pp. 171–181, 2023.

[9] G. Geetha and M. S. Kumar, "Asymmetric key cipher based on non-linear dynamics," in *Proc. First International Conference on Emerging Trends in Engineering and Technology*, 2008.

[10] W. S. Simon and J. E. Varghese, "Chaotic Based Asymmetric and Symmetric Key Encryption Using Augmented Lorenz Equations," *International Journal of Scientific Research and Management (IJSRM)*, vol. 3, no. 11, pp. 3762–3769, 2015.

[11] L. Kocarev, M. Sterjev, A. Fekete, and G. Vattay, "Public-key encryption with chaos," *Chaos*, vol. 14, no. 4, 2004.

[12] L. Kocarev, J. Makraduli, and P. Amato, "Public-Key Encryption Based On Chebyshev Polynomials," *Circuits Systems and Signal Processing*, vol. 24, pp. 497–517, 2005.

[13] R. Munir, "An Improved RC4 Algorithm Based on Multi Chaotic Map for Image Encryption," in *8th International Conference on Recent Advances and Innovations in Engineering: Empowering Computing, Analytics, and Engineering Through Digital Innovation (ICRAIE 2023)*, Kuala Lumpur, Malaysia, Dec. 2023.

[14] J. Wu, X. Liao, and B. Yang, "Image Encryption Using 2D Henon-Sine Map and DNA Approach," *Signal Processing*, vol. 153, pp. 11–23, Dec. 2018.

[15] K. Y. Cheong and T. Koshiba, "More on Security of Public-Key Cryptosystems Based on Chebyshev Polynomials," *IEEE Transactions on Circuits and Systems*, vol. 54, no. 9, 2007.