Research Article

Novel Stream Ciphering Algorithm for Big Data Images Using Zeckendorf Representation

Liangshun Wu1 and Hengjin Cai1,2

1School of Computer Science, Wuhan University, Wuhan 430079, China
2Zall Research Institute of Smart Commerce, Wuhan 430010, China

Correspondence should be addressed to Hengjin Cai; hjcai@whu.edu.cn

Received 5 August 2021; Revised 30 August 2021; Accepted 9 September 2021; Published 21 October 2021

Academic Editor: Rajesh Kaluri

Copyright © 2021 Liangshun Wu and Hengjin Cai. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Big data is a term used for very large data sets. Digital equipment produces vast amounts of images every day; the need for image encryption is increasingly pronounced, for example, to safeguard the privacy of the patients’ medical imaging data in cloud disk. There is an obvious contradiction between the security and privacy and the widespread use of big data. Nowadays, the most important engine to provide confidentiality is encryption. However, block ciphering is not suitable for the huge data in a real-time environment because of the strong correlation among pixels and high redundancy; stream ciphering is considered a lightweight solution for ciphering high-definition images (i.e., high data volume). For a stream cipher, since the encryption algorithm is deterministic, the only thing you can do is to make the key “look random.” This article proves that the probability that the digit 1 appears in the midsection of a Zeckendorf representation is constant, which can be utilized to generate the pseudorandom numbers. Then, a novel stream cipher key generator (ZPKG) is proposed to encrypt high-definition images that need transferring. The experimental results show that the proposed stream ciphering method, with the keystream of which satisfies Golomb’s randomness postulates, is faster than RC4 and LSFR with indistinguishable performance on hardware depletion, and the method is highly key sensitive and shows good resistance against noise attacks and statistical attacks.

1. Introduction

The development of digital sensor technology and storage device leads to the rapid expansion of the digital image library, and all kinds of digital equipment produce vast amounts of images every day. Though image compression reduces the bandwidth, transferring compressed images alone is still not secure. Thus, how to effectively and securely transfer these images has become a hot research direction in recent years. A variety of encryption algorithms have been investigated to image cryptosystems. Most of them are based on permutation and diffusion architecture [1]. The permutation process alters the location of image pixels, and the diffusion process changes the pixel values so that a small change in one pixel can spread to almost all pixels in the entire image [2]. These two procedures are independent. Modern cryptography includes symmetric encryption, asymmetric encryption, and hash function, among which symmetric encryption is divided into two types: block ciphers and stream ciphers. Block ciphers such as DES, AES, and IDEA, are not suitable for practical image encryption because of intrinsic features of some images such as mass data capacity, strong correlation among pixels, and high redundancy [3].

A stream cipher is a symmetric key encryption where the crypto keys used to encrypt the binary image is randomly changed so that the cipher image produced is mathematically impossible to break. Also, each bit of data is encrypted with each bit of key. The random keys are changed so that it will not allow any pattern to be repeated, giving a clue to the cracker to break the cipher image. The advantage of using stream cipher is that the execution speed is higher when compared to block ciphers and has lower hardware complexity. Unlike block ciphers, stream cipher will not produce the same ciphertext even for repetitive blocks of plaintext, since the keys are changed constantly for every bit of
plaintext. Basically, in stream ciphers, for simplicity, the manner you encrypt is by bitwise XOR, and if you intend to decrypt a ciphertext, you simply do XOR once more. The exclusive or (XOR or $\oplus$) operation, which is simple to implement on hardware, gives a ray of hope for fast image encryption.

However, if multiple data are encrypted with the same key, the attacker can decrypt the data without guessing the key. For example, suppose that two strings of plaintext $P_1$ and $P_2$ are encrypted with the same key, $K$. The ciphertexts are as follows: $E_1 = P_1 \oplus K$ and $E_2 = P_2 \oplus K$. Because $E_1 \oplus E_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$, if XOR $P_1 \oplus P_2$ on both sides, then $E_1 \oplus E_2 \oplus P_1 = P_2 \oplus P_1 = P_1$. At this point, it is clear that the attacker recovered the plaintext without obtaining the key. Therefore, in stream ciphering, the difficulty of cracking depends on the randomness and unpredictability of the keystream. Alternatively, a keystream generated by a specified generator should at least “look random.” The motivation of this paper is to generate such pseudorandom keystreams to resist chosen plaintext attacks and statistical attacks.

This paper is organized as follows: Section 2 introduces preliminary knowledge. Section 3 reviews the related work. Section 4 elaborates on generating a pseudorandom keystream that satisfies Golomb’s randomness postulates. Section 5 proves the randomness of the keystream theoretically. Section 6 does some experiments. Finally, Section 7 draws the conclusion.

2. Preliminaries

2.1. Golomb’s Randomness Postulates. Golomb’s randomness postulates [4] defines the requisite properties to be sufficiently randomly looking. Those properties are given as follows: the runs of 0’s are called “gaps”; runs of 1’s are called “blocks”.

(1) In a cycle, the number of 1’s differs from that of 0’s by at most 1

(2) At least half the runs have length 1, at least one-fourth have length 2, at least one-eighth has length 3, and so forth. Moreover, for each of these lengths, there are (almost) equally many gaps and blocks. In other words, the number of any possible $n$-runs is approximately equal to $\Lambda/2^n$, where $\Lambda$ denotes the length of the keystream.

(3) The autocorrelation function $\Gamma(\tau)$:

$$
\Gamma(\tau) = \sum_{i=1}^{\Lambda} e_i e_{i+\tau} = \begin{cases} 
\frac{\Lambda}{2}, & \tau = 0, \\
\frac{\Lambda}{4}, & 0 < \tau < \Lambda 
\end{cases}
$$

2.2. Zeckendorf Representation. It is known from Zeckendorf’s theorem [5] that each nonnegative integer can be addressed as a sum of distinct Fibonacci numbers. For instance, 17 is the sum of the 7th, 4th, and 1st Fibonacci numbers, viz. $17 = F_7 + F_4 + F_1$. Every nonnegative integer $N$ admits a representation:

$$
M = F_{m_1} + F_{m_2} + \cdots + F_{m_r},
$$

with $m_1 > m_2 > \cdots > m_r$, and as usual, $F_0 = 0$, $F_1 = 1$, and $F_{n+2} = F_{n+1} + F_n$ for all $n \geq 0$. We call this a Zeckendorf representation or $F$-addend representation of $N$. It is convenient to write this representation as a word/sequence $\{e_{L-1}, e_{L-2}, \ldots, e_0\}$ of length $L$ with each $e_i$ oscillating over the alphabet $\{0, 1\}$, where 1 indicates the respective Fibonacci addend appears in the sum, and 0 otherwise. Let

$$
S_L(N) = \{e_{L-1}, e_{L-2}, \ldots, e_0\},
$$

be the aforementioned Zeckendorf representation, for instance, $17 = F_7 + F_4 + F_1$; then,

$$
S_{\theta}(17) = \{0, 1, 0, 0, 1, 0, 0, 1, 0\}.
$$

If imposing the additional requirement that consecutive 1 are not allowed (viz. $m_i \geq m_{i+1} + 2$) and $e_i$ cannot be ‘1’ ($F_1 = F_2 = 1$, provided only $F_2 = 1$ admissible), then we obtain the canonical version of the definition [6–9]. Such a “canonical Zeckendorf representation” always exists and is unique [5].

3. Related Work

3.1. Stream Ciphers and PRNG. Stream cipher is a symmetric key cryptography in which the key is randomly altered in a way that the cipher image created is mathematically impossible to break. The benefit of using stream cipher is that when it is compared to block ciphers, the execution speed is higher and has less hardware complexity. Unlike block ciphers, stream ciphers, even for repetitive blocks of plain text, will not generate the same ciphertext, since the keys are changed constantly for every element of plaintext [10].

Image encryption using some of the existing standard stream cipher methods such as RC4 and Vernam cipher methods have drawbacks. The RC4 algorithm is vulnerable to analytic attacks of the state table. In every 256 keys, there can be a weak key [11]. These keys are identified by cryptanalysis that is able to find circumstances under which one of more generated bytes are strongly correlated with a few bytes of the key [12]. Also, the same sequence of keys is repeated which would enable the hacker to break the ciphertext. Also, the first three words of the secret key can be found, and by iteration, each word of the key used in RC4 can be obtained. The Vernam cipher considered a perfect cipher is a type of one-time pad cipher. The drawback in this method is the need for the unlimited number of keys and the distribution of large number of random keys becomes a problem [13]. Recently, the use of a chaotic system in
4. The Proposed Method

4.1. Probability Structure of Zeckendorf Representation. Suppose there exist \( m \) ones in a canonical Zeckendorf representation, denoted as \( \text{SL}_m \), it is known from [15] that the quantity of \( \text{SL}_m \) is given by

\[
N_{\text{SL}_m} = \begin{cases} 
\binom{L - m + 1}{m}, & 0 \leq m \leq \left\lfloor \frac{L + 1}{2} \right\rfloor, \\
0, & m > \left\lfloor \frac{L + 1}{2} \right\rfloor,
\end{cases}
\]

where \( \left\lfloor \cdot \right\rfloor \) is the ceiling function.

Let \( N_{\text{SL}_m}(k) \) be the number of representations that own 1 in the \( k \)th position. Filipponi and Wolfowicz [24] proves the fact that

\[
N_{\text{SL}_m}(k) = N_{\text{SL}_m}(L - k + 1),
\]

where \( 1 \leq k \leq \left\lfloor (L + 1)/2 \right\rfloor \), or

\[
N_{\text{SL}_m}(k) = \frac{1 - (-1)^k}{2} \binom{L - m - k + 1}{m - k} + \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{L - m - 2i - 1}{m - 2i - 1}.
\]

Resting on the assumption [25]:

\[
\begin{vmatrix} a \\ -b \end{vmatrix} = 0.
\]

It can be easily inferred from (7) that the addend disappears for \( i > m + 1 \). And, from (6), the following are found.

**Theorem 1.** \( N_{\text{SL}_m}(k) = N_{\text{SL}_m}(m) \), and it is constant for any \( m < k < L - m + 1 \).

**Proof.** By using (7), we compute

\[
\begin{align*}
N_{\text{SL}_0}(k) &= 0, \\
N_{\text{SL}_1}(k) &= 1, \\
N_{\text{SL}_2}(k) &= \begin{cases} 
L - 2, k \in \{1, L\}, \\
L - 3, k \notin \{1, L\},
\end{cases}
\end{align*}
\]

and using (8), we get

\[
N_{\text{SL}_m}(m) = \frac{1 - (-1)^m}{2} \binom{L - m - m + 1}{m - m} + \sum_{i=0}^{\lfloor (m-2)/2 \rfloor} \binom{L - m - 2i - 1}{m - 2i - 1}.
\]
then

\[ N_{2m-1,m}(m) = \begin{cases} 1,2|m, \\ 0,2|m, \end{cases} \]

\[ N_{2m,m}(m) = \left\lfloor \frac{m+1}{2} \right\rfloor, \]

\[ N_{2m+1,m}(m) = \begin{cases} \frac{(m+1)^2}{4} - 2|m, \\ \frac{m(m+2)}{4} - 2|m. \end{cases} \]  \hfill (12)

Therefore, the probability that the \( k \)th position of a Zeckendorf representation containing \( m \) 1s locates the digit 1 is

\[ \Pr_{L,m}(k) = \frac{N_{L,m}(m)}{N_{L,m}}. \]  \hfill (13)

In a similar fashion of (10) and (12), we have

\[ \Pr_{L,m}(1) = \Pr_{L,m}(L) = \frac{m}{L-m-1}, \]

\[ \Pr_{L,m}(2) = \Pr_{L,m}(L-1) = \frac{m(L-2m+1)}{(L-m-1)(L-m)}, \]

\[ \Pr_{2m-1,m}(m) = \begin{cases} 1,2|m, \\ 0,2|m, \end{cases} \]

\[ N_{2m,m} = \begin{cases} \frac{1}{2}, \\ \frac{m}{2m+2}, \end{cases} \]

\[ N_{2m+1,m} = \begin{cases} \frac{m+1}{2}, \\ \frac{m}{2m+2}, \end{cases} \]  \hfill (14)

Let \( L = 30, m = 9 \), then, the value of \( \Pr_{L,m}(k) \) is shown as Figure 1, where the straight line in the midsection means that the probability is constant.

4.2. Pseudorandom Keystream Generation. The following procedures could generate a reasonably satisfying keystream \( C \):

Suppose that both the sender and the receiver know a pair of keys \((e_1, e_2)\), each of which consists of three integers:

\[ \begin{cases} e_1 = (a_n, c_n, \mu_n), \\ e_2 = (a_m, c_m, \mu_m), \end{cases} \]  \hfill (15)

where \( \mu_n \) and \( \mu_m \) are primes of the same order of magnitude. There exist pseudorandom integral sequences

\[ \{ N = (N_1, N_2, \ldots, N_H), \]

\[ \{ M = (M_1, M_2, \ldots, M_H), \]  \hfill (16)

with starting values \((N_0, M_0)\) satisfying

\[ \begin{cases} N_0 > 0, & a_n, c_n < \mu_n, \\ M_0 > 0, & a_m, c_m < \mu_m, \end{cases} \]  \hfill (17)

and each item of \( \mathbb{N} \) and \( \mathbb{M} \) is obtained by the algorithm described in [26] that

\[ \begin{cases} N_{h+1} = (a_n N_h + c_n) \mod \mu_n, & 0 \leq h \leq H - 1, \\ M_{h+1} = (a_m N_h + c_m) \mod \mu_m, & 0 \leq h \leq H - 1, \end{cases} \]  \hfill (18)

where \( H \) is decided by the message length. The integers \( N_i, M_i \ (i = 1, 2, \ldots, H) \) are converted into canonical Zeckendorf representations:

\[ \begin{cases} u_L(N_i) = (n_1, n_2, \ldots, n_L), \\ u_L(M_i) = (m_1, m_2, \ldots, m_L). \end{cases} \]  \hfill (19)

Then, we carry out the bitwise logical addition (OR) on their midsection (where the probability is constant) in this way acquiring \( C_i \) of length \( t \):

\[ C_i = \{ e_1, e_2, \ldots, e_t \}, \]  \hfill (20)
The sequence $C$ of length $Ht$:

$$C = \{C_1, C_2, \ldots, C_H\}.$$  

By juxtaposing $C$, we finally obtain the keystream sequence $C$ of length $Ht$:

$$C = \{C_1, C_2, \ldots, C_H\}.$$  

The sequence $C$ is exactly what we need. Just for the sake of convenience narration, we refer to this kind of pseudorandom keystream generation algorithm as “ZPKG” hereinafter, and the pseudocode is shown as Algorithm 1.

**Algorithm 1: Pseudorandom keystream generation algorithm.**

```plaintext
Require: A pair of key, $c_1 = (a_{n}, \mu_{n}), c_2 = (a_{m}, \mu_{m})$; message length, $H$; 
Ensure: Keystream, $C = \{C_1, C_2, \ldots, C_H\}$; 
1: //initialize; 
2: $N_n \leftarrow \text{Random}();$ 
3: $M_o \leftarrow \text{Random}();$ 
4: for $h = 1$ to $H$ do 
5:  
6:  Generate integral sequences 
7:  
8:  $N_{h+1} \leftarrow (a_{n} \cdot N_{h} + c_{n}) \mod \mu_{n};$ 
9:  $M_{h+1} \leftarrow (a_{m} \cdot M_{h} + c_{m}) \mod \mu_{m};$ 
10: /*************************************************************/ 
11: Convert into Zeckendorf representations 
12: /*************************************************************/ 
13: $S_c(N_h) \leftarrow \text{Encode}(N_h);$ 
14: $S_c(M_h) \leftarrow \text{Encode}(M_h);$ 
15: /*************************************************************/ 
16: Intercept midsection 
17: /*************************************************************/ 
18: for $u = S_c(N_h), S_c(M_h)$ do 
19:  //Count the number of 1 in sequence; 
20:  $m \leftarrow u.\text{Count}(1);$ 
21:  //Determine the middle-section; 
22:  $L_{start} \leftarrow m;$ 
23:  $L_{end} \leftarrow L - m + 1;$ 
24:  //Intercept; 
25:  $u \leftarrow u[L_{start}, L_{end}];$ 
26: end for 
27: /*************************************************************/ 
28: Bitwise OR 
29: /*************************************************************/ 
30: $s \leftarrow S_c(N_h) \text{ OR } S_c(M_h);$ 
31: /*************************************************************/ 
32: Take random piece (of length $t$) 
33: /*************************************************************/ 
34: $U_L = (5(L + 2) - 8 - [5(L + 2)^2 + 4]^{1/2} / 10) + 1;$ 
35: $t = L - 2U_L + 2;$ 
36: $C_h = \{c_1, c_2, \ldots, c_t\} \leftarrow \text{RandomSelect}(s)$ 
37: end for 
38: Return $C = \{C_1, C_2, \ldots, C_H\};$
```

5. Randomness Analysis

Let $F_n$ be the greatest Fibonacci number no greater than $N_i$, then the length of the shortest Zeckendorf representation $S_L(N_i)$ will be $L = n - 1$. It can be proved that

$$L = \left\lfloor \log_\Phi \sqrt{5} \left( N_i + \frac{1}{2} \right) \right\rfloor - 1,$$  

where $\Phi = (1 + \sqrt{5})/2$ denotes the golden ratio. $U_L$ of 1’s in $S_L(N_i)$ is most likely to be [15]:

$$U_L = \frac{5(L + 2) - 8 - [5(L + 2)^2 + 4]^{1/2}}{10} + 1.$$
The probability that a digit '1' lies in the $k$th position in the midsection of $S_L(N_i)$ is

$$p(1) = \Pr_{L,U_i}(N_i) = \frac{N_{L,U_i}(U_i)}{N_{L,U_i}}. \quad (25)$$

Then, the probability that a '0' lies in the $k$th position in both $S_L(N_i)$ and $S_L(M_i)$ is readily given as below:

$$\Pr(0) = p^2(0) = \frac{\Phi^2}{5} \approx 0.524, \quad (26)$$

where

$$p(0) = 1 - p(1) \approx \frac{(\Phi + 1)}{(\Phi + 2)} \approx 0.724. \quad (27)$$

Similarly, it holds for $S_L(M_i)$.

It draws from [15] that as $L$ approaches infinity, even in this case, $L > 25$ would be enough, $p(1)$ is expected to approach the limit of $1/(\Phi + 2) \approx 0.2764$ (see Figure 2).
is blamed. Therefore, we have followed by ‘and’ going to assess the probabilities and accounts, to be so all around full filled.

\[ \Pr(1) = 1 - \Pr(0) = 1 - p^2(0) \approx 0.476. \]  

(28)

From (26) and (28), it follows that, for \( L > 25 \), Golomb’s first postulate is enough fulfilled.

Golomb’s second postulate does not seem, by all accounts, to be so all around fulfilled. In this paper, we are going to assess the probabilities \( \Pr(00), \Pr(01), \Pr(10), \) and \( \Pr(11) \) of any conceivable pair in \( C \).

First, we think about the probabilities \( \Pr(00), \Pr(01), \) and \( \Pr(10) \) and see that ‘1’ fundamentally preexists or is followed by ‘0’—the fact that there is no pair (11) at all that is blamed. Therefore, we have

\[ p(01) = p(10) = p(1) = \frac{1}{\Phi + 2}, \]  

(29)

\[ p(00) = 1 - (p(01) + p(10)) = 1 - 2p(1) = \frac{\Phi}{\Phi + 2}. \]  

(30)

We can apply some bitwise logical additions (OR or +) to get each pair of \( C \):

\[ (00) = (00) + (00), \]  

(31)

\[ (1) = (01) + (00) \text{ or } (00) + (10) \text{ or } (01) + (10) \]

\[ (2) = (10) + (00) \text{ or } (00) + (10) \text{ or } (10) + (10) \]

\[ (3) = (11) + (01) \text{ or } (01) + (10) \]
for relatively large $L_1$ ($L_1 = 47$); in other words, the experimental estimations are near theoretical calculation when $L_1 \rightarrow \infty$.

We extract a segment of length $\Lambda' = 1,000$ randomly from $C$ and then compute the estimation of $\Gamma(\tau) = \sum_{i=1}^{\Lambda'} c_i$ for $\tau = 1, 2, \ldots, \Lambda' - 1$, and unquestionably:

$$
\Gamma(0) = 0.47\Lambda',
\Gamma(1) = \Gamma(\Lambda' - 1) = 0.163\Lambda',
0.2\Lambda' \leq \Gamma(\tau) \leq 0.247\Lambda' \left( \bar{T}(\tau) = 0.223\Lambda' \right),
2 \leq \tau \leq \Lambda' - 2,
$$

where $\bar{T}(\tau) = (1/\Lambda')\sum_{r=0}^{\Lambda'-1} \Gamma(\tau)$.

A few more cases were acquired by alternating the parameters $N_0$, $M_0$, $e_1$, and $e_2$ rendered insignificant differences from the preceding cases, which prove that the ZPKG algorithm satisfies the Golomb randomness postulates.

6.1.2. NIST-800-22 Statistical Testing. NIST-800-22 [27] is a statistical test suite for random and pseudorandom number generators for cryptographic applications. This test standard was enacted by the Information Technology Laboratory.
(ITL) at the National Institute of Standards and Technology (NIST). The test suite describes 16 statistical tests, including the longest run test, cumulative sums, and the linear complexity test, which are useful in detecting deviations of a binary sequence from randomness. The \( p \) value summarizes the strength of the evidence against the null hypothesis in each statistical test. If \( p \) value \( \geq \alpha \) (level of significance), then the null hypothesis is accepted; i.e., the sequence appears to be random. If \( p \) value \( < \alpha \), the null hypothesis is rejected; i.e., the sequence appears nonrandom. Typically, \( \alpha \) is chosen in the range \([0.001, 0.01]\). Common values of \( \alpha \) in cryptography are about 0.01 based on the NIST-800-22 test standard.

6.2. Performance Evaluation. \( m \)-sequence based on linear feedback shift register (LFSR) is a widely used keystream generator for its long period, good statistical characteristics, easy to be analyzed by algebraic methods, and adapted for hardware implementation. Another type is word-based stream ciphers, for example, RC4 [28, 29]. RC4 has a variable key length and is based on the word-driven operation using random permutations. Unlike LFSR, RC4 works better with software implementation. RC4 consists of two parts: PRGA algorithm, which is for a pseudorandom number generator, and KSA algorithm, which is for key generation. RC4 is extensively used in the secure sockets protocol/transport layer security (SSL/TLS) and WEP protocols, part of the IEEE802.11 wireless LAN standard.

In this section, the ZPKG algorithm, RC4 algorithm, and LFSR algorithm are successfully applied to the encryption of more than 50 images of CVG-UGR test image set, including gray image, biometric image, medical image, and magnetic resonance image (MRI). The experimental simulation platform is FPGA, and the simulation software is ModelSim SE-64 10.4.

6.2.1. Hardware Depletion. As shown in Table 3, the ZPKG circuit employs significantly more logic gates and registers than RC4 and LFSR; it occupies the same RAM as LFSR but is slightly higher than RC4. However, the I/O pins of the ZPKG circuit are only 1/3 of RC4, slightly more than that of LFSR. The number of PLLs is similar to that of RC4 but higher than LFSR. This shows that the ZPKG and RC4 circuits have different priorities regarding the disposal of hardware resources; ZPKG and RC4 occupied more hardware resources than LFSR. From a power distribution perspective, they are all primarily based on static power. ZPKG has the highest I/O power other than dynamic power (see Figure 3 and Table 4).

However, as shown in Table 3, the number of ZPKG I/O pins is considerably lower than RC4, implying that ZPKG requires frequent I/O operations. In addition, the clock power consumption of RC4 is much higher than that of ZPKG and LFSR, which indicates that RC4 requires more clock cycles, which suggests that the generation speed of a pseudorandom keystream of RC4 is the slowest.

6.2.2. Key Generation Speed. Under crystal vibration frequency of 50 MHz, the ZPKG circuit spends 4670 ns to generates a 64-bit pseudorandom keystream, while that of the RC4 is 9790 ns, and that of the LFSR is 25500 ns (see Table 5). In other words,
Figure 5: Continued.
Figure 5: Grayscale images encryption and decryption using ZPKG algorithm. (a–d) Original. (e–h) Encrypted images. (i–l) Decrypted images (SSIM = 1).
Figure 6: Continued.
ZPKG is approximately one time faster than RC4; both ZPKG and RC4 are much slower than LFSR.

Twenty simulations were completed, each generating a pseudorandom 64-bit key. Figure 4 presents the statistics, suggesting that the results are stable with the simulations.

6.3. Security Analysis. Security is important not only for the encrypted objects but also for the encryption algorithms themselves.

In what follows, we discuss some security issues of the ZPKG algorithm, such as scrambling effect, statistical...
Figure 7: Continued.
histogram analysis, key sensitivity testing, robustness, and noise attacks.

6.3.1. Scrambling Effect. Figure 5 shows the results of ZPKG algorithm encrypting and decrypting different types of gray-scale images. The encrypted images are visually close to noise images. The structural similarity (SSIM) index is a quantitative assessment method for measuring the similarity between two images [29]; it reflects whether the original images are completely reconstructed or not. A value 1 of
the SSIM index indicates that two measured images are identical. SSIM is computed as

$$\text{SSIM} = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X) \text{Var}(Y)}},$$  

where

$$\text{Cov}(X, Y) = N \sum_{i=1}^{N} x_i y_i - \left( \sum_{i=1}^{N} x_i \right) \left( \sum_{i=1}^{N} y_i \right),$$

$$\bar{X} = N \sum_{i=1}^{N} x_i^2 - \left( \sum_{i=1}^{N} x_i \right)^2,$$

$$\bar{Y} = N \sum_{i=1}^{N} y_i^2 - \left( \sum_{i=1}^{N} y_i \right)^2,$$

where $x_i$ represents the gray value of the $i$th pixel of the first image, and $y_i$ represents the gray value of the $i$th pixel of the second image.

Figures 5(i)–5(l) show the grayscale images decrypted by ZPKG, and its SSIM index equals to 1, which proves the correctness of the ZPKG algorithm.

The 3D images such as color images contain several 2D data matrices called 2D components. Color images, for example, contain three color channels called R, G, and B. Each color channel is a 2D component. In this manner, the 3D images can be considered the combination of several 2D images. The 3D image encryption can be accomplished by encrypting all its 2D components one by one. Figure 6 shows three examples of the ZPKG algorithm encrypting color images. The encrypted images (Figures 6(d)–6(f)) look like noise images visually.

6.3.2. Histogram Analysis. An image histogram is a graphic representation of the pixel intensity distribution of an image. To overcome statistic attacks, the encrypted image should have a histogram with random behavior and uniform distribution.

The encrypted image histograms using the ZPKG algorithm, LFSR algorithm, and RC4 algorithm are completely different from the original image (see Figure 7). Nevertheless,
Figure 9: Continued.
the encrypted image of RC4 still retains some visual information of the original image, and the intensity distribution of the corresponding histogram is uneven. By contrast, the encrypted images generated by the LFSR and ZPKG algorithms follow a nearly uniform distribution, indicating that the ZPKG algorithm has better performance against statistical attacks than RC4.

6.3.3. Key Sensitivity. We flip one bit of keystream, then decrypt images with flipped key. We notice that the decoded picture is in a state of chaos (see Figure 8) and deviates from the original image. The ZPKG algorithm shows good key sensitivity.

6.3.4. Robustness. The communication and networking channels are generally in the presence of different types of noise. To test the robustness of the ZPKG algorithm against noise attacks, the salt and pepper noise with density 0.05 is added to the encrypted images. We then try to reconstruct the original image from these noised encrypted images. The SSIM index is used to quantitatively evaluate the similarity between the reconstructed images and the original images. The results are shown in Figure 9. The SSIM index of ZPKG (0.9021) is higher than that of RC4 (0.7037 and 0.4548) and LFSR (0.5068), so we can say ZPKG is more robust when facing noise attacks.

7. Conclusion

Stream ciphers cannot really work without the keystream randomness. We proved that the probability of occurrence of the number 1 in the middle part of Zeckendorf coding is constant, which can generate pseudorandom numbers. The pseudorandom numbers generated by the proposed algorithm satisfy the Golomb randomness hypothesis. Experimental results show that our method is three times faster than the RC4 and LSFR algorithms, has no significant difference in hardware occupation, and has high key sensitivity and good resistance to noise attacks and statistical attacks. There is no doubt that our research has its limitations, for example, the lack of theoretical analysis of the characteristics of cryptography, and the lack of comparison with recent research such as chaotic stream ciphering algorithms. We will spend more time devoting on theoretical research on Zeckendorf representation in the future. Furthermore, more experimental investigations are needed to evaluate the performance of ZPKG by comparing it with other stream ciphers such as RC4A, VMPC, and Spritz.

Data Availability

The CVG-UGR test images data used to support the findings of this study have been deposited in the repository (https://ccia.ugr.es/cvg/dbimagenes/).

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This work is sponsored by the National Natural Science Foundation of China under grant number 61832014.

References


