Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 4637876, 19 pages
https://doi.org/10.1155/2021/4637876

Research Article

Novel Stream Ciphering Algorithm for Big Data Images Using Zeckendorf Representation

Liangshun Wu[1] **and Hengjin Cai**[1,2]

[1]*School of Computer Science, Wuhan University, Wuhan 430079, China*
[2]*Zall 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

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 data, P_1 and P_2, are encrypted using the same key, K. The ciphertexts are as follows: $E_1 = P_1 \oplus K$ and $E_2 = P_2 \oplus K$. Because $E_1 \oplus E_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2 \oplus (K \oplus K) = P_1 \oplus P_2$, if XOR P_2 on both sides, then $E_1 \oplus E_2 \oplus P_2 = P_1 \oplus P_2 \oplus P_2 = P_1$. 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 random 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 Λ denotes the length of the keystream

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

$$\Gamma(\tau) = \sum_{i=1}^{\Lambda} c_i c_{i+\tau} = \begin{cases} \dfrac{\Lambda}{2}, & \tau = 0, \\ \dfrac{\Lambda}{4}, & 0 < \tau < \Lambda \end{cases} \tag{1}$$

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:

$$M = F_{m_1} + F_{m_2} + \cdots + F_{m_r}, \tag{2}$$

with $m_1 > m_2 > \cdots \geq m_r$, 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 $\{\varepsilon_{L-1}, \varepsilon_{L-2}, \cdots, \varepsilon_0\}$ of length L with each ε_i oscillating over the alphabet $\{0, 1\}$, where 1 indicates the respective Fibonacci addend appears in the sum, and 0 otherwise. Let

$$S_L(N) = \{\varepsilon_{L-1}, \varepsilon_{L-2}, \cdots, \varepsilon_0\}, \tag{3}$$

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

$$S_9(17) = \{0, 1, 0, 0, 1, 0, 0, 1, 0\}. \tag{4}$$

If imposing the additional requirement that consecutive 1 are not allowed (viz. $m_i \geq m_{i+1} + 2$) and ε_1 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

cryptography to encrypt images has emerged for its random characteristics [14].

As the core component of stream ciphers, the generation of random numbers is essential. There are two basic types of generators used to produce random sequences: random number generators (RNGs) and pseudorandom number generators (PRNGs). For cryptographic applications, both of these generator types produce a stream of zeros and ones that may be divided into substreams or blocks of random numbers. A random bit sequence could be interpreted as the result of the flips of an unbiased "fair" coin with sides that are labeled "0" and "1." Obviously, the use of unbiased coins for cryptographic purposes is impractical. An RNG considers a nondeterministic source (i.e., the entropy source). The source typically consists of some physical quantity, such as the noise in an electrical circuit, the timing of user processes (e.g., keystrokes or mouse movements), or the quantum effects in a semiconductor. The outputs of an RNG may be used directly as a random number or may be fed into a PRNG. However, producing high-quality random numbers may be so time-consuming. Inputs to PRNGs are called seeds. The outputs of a PRNG are typically deterministic functions of the seed, which is the origin of the term "pseudorandom." Ironically, pseudorandom numbers often appear to be more random than RNGs, because a series of transformations can eliminate statistical autocorrelations between input and output [10].

3.2. Applications of Zeckendorf Representation. Zeckendorf representation works well in certain situations. For example, Leroy et al. [15] count the number of distinct (scattered) subwords occurring in a given word. More precisely, it considers the generalization of the Pascal triangle to the binomial coefficients of words and the Zeckendorf representation counting the number of positive entries on each row. Epifanio et al. [16] proved that Zeckendorf representation has deep connections with the Sturmian graph, and Bernat [17] connected Zeckendorf representation with continued fractions.

In addition, the research of stream ciphers that resorts to Zeckendorf representation has long been done. For example, feedback with carry shift registers (FCSRs) plays a vital role in the hardware design of stream ciphers besides LFSRs. Galois representation is often considered the first choice for FCSRs, howbeit, recently, a new representation that generalizes both Galois and Zeckendorf representations for FCSR automata was presented [18]. It is immune to previous attacks and can dramatically improve internal diffusion. Later, Lin [19] further improved the aforementioned FCSR circuit.

Similarly, the U-Quark hash function with FCSRs of Zeckendorf representation [20]. Fish (Fibonacci shrinking), a fast software stream cipher, was proposed to achieve solid performance simulated on an Intel 486 processor [21]. Nevertheless, these researches do not focus on the generation of keystream of the stream cipher at the software level but the hardware level. Our research is suggested adding a small stone to the wall of the application of Zeckendorf representation. Recently, several studies apply Zeckendorf representation in blockchain and big data encryption [22, 23].

4. The Proposed Method

4.1. Probability Structure of Zeckendorf Representation. Suppose there exist m ones in a canonical Zeckendorf representation, denoted as $S_{L,m}$, it is known from [15] that the quantity of $S_{L,m}$ is given by

$$N_{L,m} = \begin{cases} \binom{L-m+1}{m}, & 0 \le m \le \left\lceil \frac{L+1}{2} \right\rceil, \\ 0, & m > \left\lceil \frac{L+1}{2} \right\rceil, \end{cases} \tag{5}$$

where $\lceil \cdot \rceil$ is the ceiling function.

Let $N_{L,m}(k)$ be the number of representations that own 1 in the kth position. Filipponi and Wolfowicz [24] proves the fact that

$$N_{L,m}(k) = N_{L,m}(L-k+1), \tag{6}$$

$$N_{L,m}(k) = \sum_{i=0}^{k-1} (-1)^i \binom{L-m-i}{m-1-i}, \tag{7}$$

where $1 \le k \le \lceil (L+1)/2 \rceil$, or

$$N_{L,m}(k) = \frac{1-(-1)^k}{2} \binom{L-m-k+1}{m-k} + \sum_{i=0}^{\lceil (k-2)/2 \rceil} \binom{L-m-2i-1}{m-2i-1}, \tag{8}$$

resting on the assumption [25]:

$$\binom{a}{-|b|} = 0. \tag{9}$$

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

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

Proof. By using (7), we compute

$$\begin{aligned} N_{L,0}(k) &= 0 \\ N_{L,1}(k) &= 1, \\ N_{L,2}(k) &= \begin{cases} L-2, k \in \{1, L\}, \\ L-3, k \notin \{1, L\}, \end{cases} \end{aligned} \tag{10}$$

and using (8), we get

$$\begin{aligned} N_{L,m}(m) &= \frac{1-(-1)^m}{2} \binom{L-m-m+1}{m-m} \\ &\quad + \sum_{i=0}^{\lceil (m-2)/2 \rceil} \binom{L-m-2i-1}{m-2i-1}, \end{aligned} \tag{11}$$

then

$$N_{2m-1,m}(m) = \begin{cases} 1, 2\nmid m, \\ 0, 2\mid m, \end{cases}$$
$$N_{2m,m}(m) = \left\lceil \frac{m+1}{2} \right\rceil, \tag{12}$$
$$N_{2m+1,m}(m) = \begin{cases} \frac{(m+1)^2}{4}, 2\nmid m, \\ \frac{m(m+2)}{4}, 2\mid m. \end{cases}$$

□

Therefore, the probability that the kth position of a Zeckendorf representation containing m 1 s locates the digit 1 is

$$\Pr_{L,m}(k) = \frac{N_{L,m}(k)}{N_{L,m}}. \tag{13}$$

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

$$\Pr_{L,m}(1) = \Pr_{L,m}(L) = \frac{m}{L-m-1},$$
$$\Pr_{L,m}(2) = \Pr_{L,m}(L-1) = \frac{m(L-2m+1)}{(L-m-1)(L-m)},$$
$$\Pr_{2m-1,m}(m) = \begin{cases} 1, 2\nmid m, \\ 0, 2\mid m, \end{cases}$$
$$N_{2m,m} = \begin{cases} \frac{1}{2}, & 2\nmid m, \\ \frac{m}{2m+2}, & 2\mid m, \end{cases} \tag{14}$$
$$N_{2m+1,m} = \begin{cases} \frac{m+1}{2m+4}, & 2\nmid m, \\ \frac{m}{2m+2}, & 2\mid m. \end{cases}$$

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:

$$\begin{cases} e_1 = (a_n, c_n, \mu_n), \\ e_2 = (a_m, c_m, \mu_m), \end{cases} \tag{15}$$

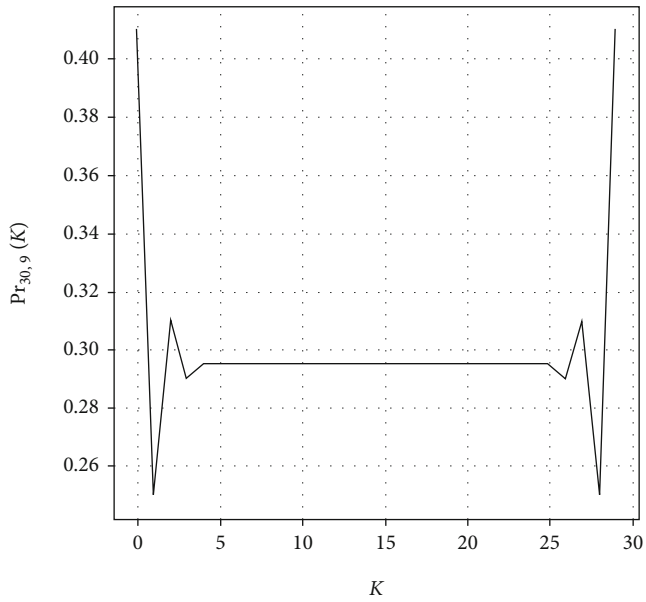


Figure 1: The diagram of $\Pr_{30,9}(k)$.

where μ_n and μ_m are primes of the same order of magnitude. There exist pseudorandom integral sequences

$$\begin{cases} \mathbb{N} = (N_1, N_2, \cdots, N_H), \\ \mathbb{M} = (M_1, M_2, \cdots, M_H), \end{cases} \tag{16}$$

with starting values (N_0, M_0) satisfying

$$\begin{cases} N_0 > 0, \quad a_n, c_n < \mu_n, \\ M_0 > 0, \quad a_m, c_m < \mu_m, \end{cases} \tag{17}$$

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

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

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

$$\begin{cases} u_L(N_i) = (n_1, n_2, \cdots, n_L), \\ u_L(M_i) = (m_1, m_2, \cdots, m_L). \end{cases} \tag{19}$$

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:

$$\mathbf{C}_i = \{c_1, c_2, \cdots, c_t\}, \tag{20}$$

Require: A pair of key, $e_1 = (a_n, c_n, \mu_n)$, $e_2 = (a_m, c_m, \mu_m)$; message length, H;
Ensure: Keystream, $C = \{C_1, C_2, \cdots, C_H\}$;
//initialize;
$N_0 \longleftarrow Random()$;
$M_0 \longleftarrow Random()$;
for $h = 1$ to H **do**
/**********
Generate integral sequences
**********/
$N_{h+1} \longleftarrow (a_n * N_h + c_n) \bmod \mu_n$;
$M_{h+1} \longleftarrow (a_m * M_h + c_m) \bmod \mu_m$;
/**********
Convert into Zeckendorf representations
**********/
$S_L(N_h) \longleftarrow \text{Encode}(N_h)$;
$S_L(M_h) \longleftarrow \text{Encode}(M_h)$;
/**********
Intercept midsection
**********/
for $u = S_L(N_h), S_L(M_h)$ **do**
//Count the number of 1 in sequence;
$m \longleftarrow u.\text{Count}(1)$;
//Determine the middle-section;
$L_{\text{start}} \longleftarrow m$;
$L_{\text{end}} \longleftarrow L - m + 1$;
//Intercept;
$u \longleftarrow u[L_{\text{start}}, L_{\text{end}}]$;
end for
/**********
Bitwise OR
**********/
$s \longleftarrow S_L(N_h)$ OR $S_L(M_h)$
/**********
Take random piece (of length t)
**********/
$U_L = (5(L+2) - 8 - [5(L+2)^2 + 4]^{1/2}/10) + 1$;
$t = L - 2U_L + 2$;
$C_h = \{c_1, c_2, \cdots, c_t\} \longleftarrow \text{RandomSelect}(s)$
end for
Return $C = \{C_1, C_2, \cdots, C_H\}$;

Algorithm 1: Pseudorandom keystream generation algorithm.

where t is given by (see next section for the value of U_L)

$$t = L - 2U_L + 2,$$
$$c_j = n_k + m_k, j = 1, 2, \cdots, t\,; k = U_L + j - 1. \tag{21}$$

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

$$\mathbf{C} = \{\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_H\}. \tag{22}$$

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.

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

$$L = \left\lfloor \log_{\Phi} \sqrt{5} \left(N_i + \frac{1}{2} \right) \right\rfloor - 1, \tag{23}$$

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]:

$$U_L = \frac{5(L+2) - 8 - \left[5(L+2)^2 + 4\right]^{1/2}}{10} + 1. \tag{24}$$

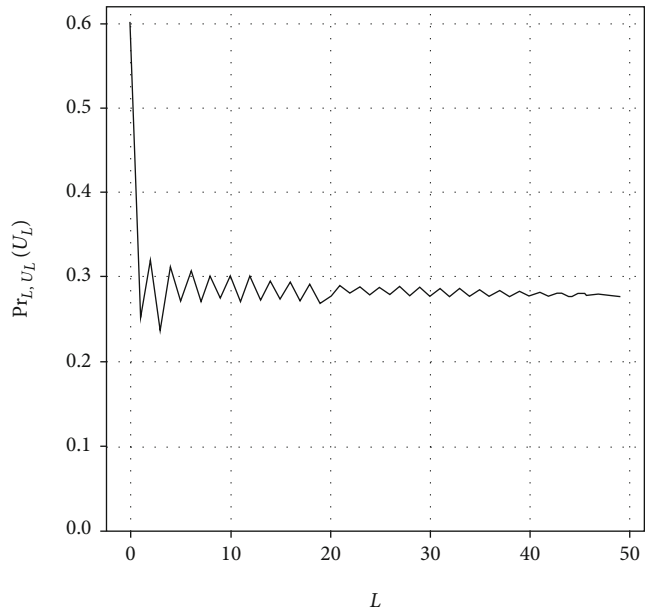


Figure 2: The diagram of $\Pr_{L,U_L}(U_L)$.

Table 1: The probability of the n-runs that appear in C ($n = 1, 2, 3, 4$).

$n = 1$		$n = 2$		$n = 3$		$n = 4$	
Runs	Probability	Runs	Probability	Runs	Probability	Runs	Probability
(0)	0.525	(00)	0.201	(000)	0.078	(0000)	0.030
(1)	0.475	(01)	0.322	(001)	0.124	(0001)	0.047
		(10)	0.322	(010)	0.225	(0010)	0.089
		(11)	0.155	(011)	0.096	(0011)	0.036
				(100)	0.124	(0100)	0.086
				(101)	0.197	(0101)	0.138
				(110)	0.096	(0110)	0.059
				(111)	0.060	(0111)	0.037
						(1000)	0.047
						(1001)	0.076
						(1010)	0.136
						(1011)	0.060
						(1100)	0.038
						(1101)	0.058
						(1110)	0.037
						(1111)	0.023

The probability that a digit '1' lies in the kth position in the midsection of $S_L(N_i)$ is

$$p(1) = \Pr_{L,U_L}(U_L) = \frac{N_{L,U_L}(U_L)}{N_{L,U_L}}. \tag{25}$$

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)

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

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

where

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

Table 2: NIST-800-22 statistical testing result of ZPKG algorithm.

Test item	Params 1	Params 2	Params 3	Params 4	Result
Approximate entropy	0.026853	0.013829	0.068205	0.034937	Pass
Block frequency	0.058378	0.870831	0.724584	0.297646	Pass
Cumulative sums	0.459642	0.069717	0.963210	0.328997	Pass
FFT	0.358795	0.919848	0.081236	0.713570	Pass
Frequency	0.435391	0.447255	0.888660	0.193601	Pass
Linear complexity	0.186537	0.203633	0.569565	0.232544	Pass
Longest run	0.359643	0.087189	0.789913	0.250387	Pass
Nonoverlapping template	0.348045	0.680967	0.106169	0.068529	Pass
Overlapping template	0.512834	0.063236	0.020689	0.490518	Pass
Random excursions	0.319514	0.181174	0.524622	0.304589	Pass
Random excursion variant	0.579380	0.177934	0.108254	0.659874	Pass
Rank	0.949536	0.648387	0.862457	0.648387	Pass
Runs	0.340097	0.086469	0.041369	0.027231	Pass
Serial test-1	0.407933	0.213432	0.648688	0.814738	Pass
Serial test-2	0.462490	0.880617	0.584615	0.512974	Pass
Maurer's universal	0.026152	0.538143	0.142680	0.600293	Pass

In this way, $\Pr(1)$ is given by

$$\Pr(1) = 1 - \Pr(0) = 1 - p^2(0) \approx 0.476. \tag{28}$$

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

$$p(01) = p(10) = p(1) \approx \frac{1}{(\Phi + 2)}, \tag{29}$$

$$p(00) = 1 - (p(01) + p(10)) = 1 - 2p(1) \approx \frac{\Phi}{(\Phi + 2)}. \tag{30}$$

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

$$(00) = (00) + (00), \tag{31}$$

(1) $(01) = (01) + (00)$ or $(00) + (10)$ or $(01) + (10)$

(2) $(10) = (10) + (00)$ or $(00) + (10)$ or $(10) + (10)$

(3) $(11) = (10) + (01)$ or $(01) + (10)$

Table 3: Hardware resource depletion comparison.

	ZPKG	RC4	LFSR
Logic elements	1110	375	205
Registers	303	80	34
Pins	37	104	21
RAM	21B	0B	20B
PLLs	32	30	0

Next, from (29), (30), and 1-4, we have

$$\begin{aligned} \Pr(00) &= p^2(00) = \frac{1}{5} = 0.2, \\ \Pr(01) = \Pr(10) &= 2p(00)p(01) + p^2(01) \approx \frac{\Phi}{5} \approx 0.324, \\ \Pr(11) &= 2p^2(01) \approx \frac{2}{5\Phi^2} \approx 0.152. \end{aligned} \tag{32}$$

6. Experiment

6.1. Randomness Test

6.1.1. Golomb's Postulate Testing. Given the initial values $e_1 = \{a_n = 29, c_n = 4{,}712{,}321{,}103, \mu_n = 4{,}500{,}000{,}013\}$, $e_2 = \{a_m = 31, c_m = 5{,}666{,}778{,}007, \mu_m = 5{,}127{,}312{,}451\}$, $N_0 = 5{,}123{,}123{,}007$, $M_0 = 4{,}901{,}976{,}445$, $H = 100$. Put them into the formula, we have $L = 47$, $U_l = 13$, and $t = 23$. Therefore, $\Lambda = Ht = 23{,}000$. The probability of n-runs is obtained by enumerating the occasions they appear in C and dividing Λ. Table 1 shows the results. The results of $n = 1$ and $n = 2$ prove that (26), (28), and (32) hold

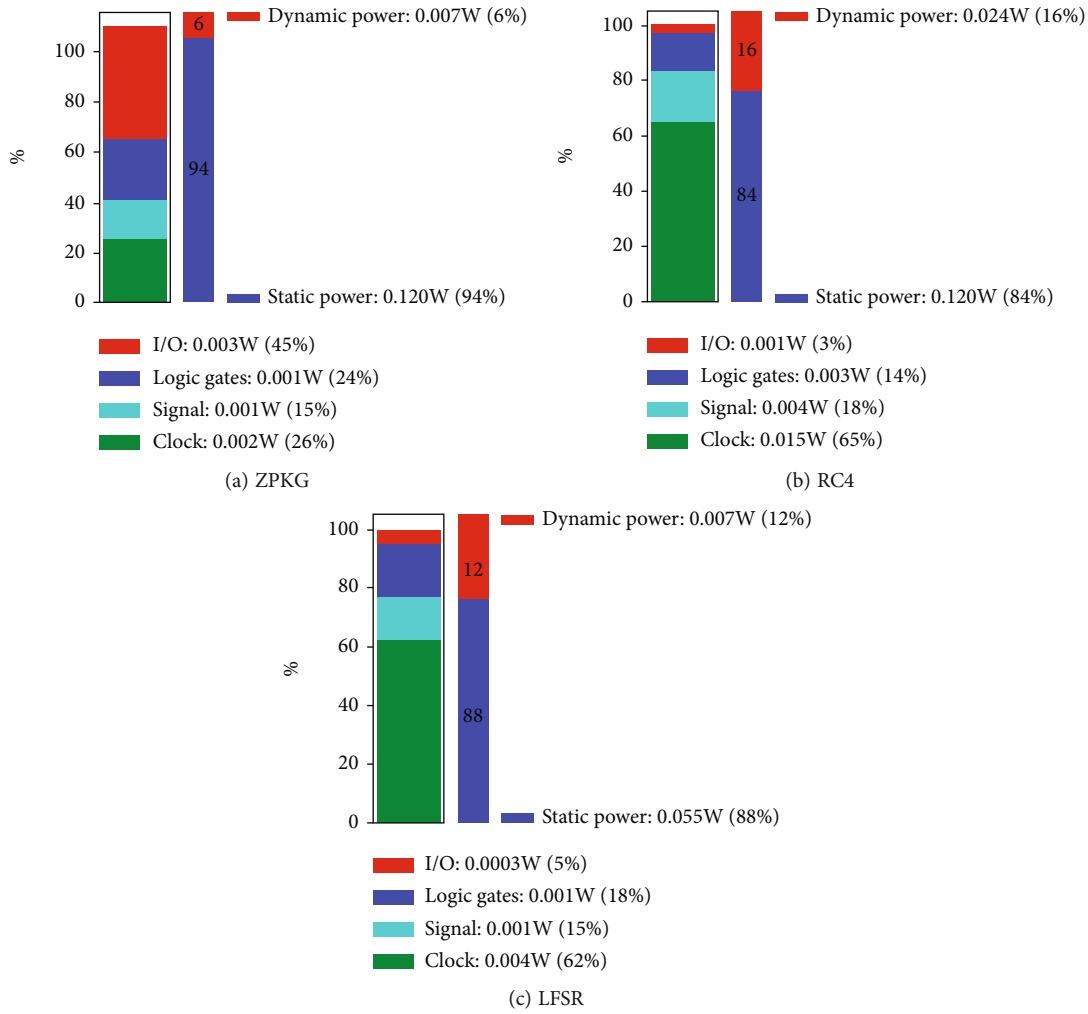


Figure 3: The power dissipation distribution.

for relatively large L ($L = 47$); in other words, the experimental estimations are near theoretical calculation when $L \longrightarrow +\infty$.

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

$$\begin{aligned} &\Gamma(0) = 0.47\Lambda', \\ &\Gamma(1) = \Gamma\left(\Lambda' - 1\right) = 0.163\Lambda', \\ &0.2\Lambda' \le \Gamma(\tau) \le 0.247\Lambda' \left(\overline{\Gamma}(\tau) = 0.223\Lambda'\right), \\ &\quad 2 \le \tau \le \Lambda' - 2, \end{aligned} \tag{33}$$

where $(\overline{\Gamma}(\tau) = (1/\Lambda')\sum_{\tau=0}^{\Lambda-1}\Gamma(\tau))$.

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.

Table 4: The power dissipation comparison.

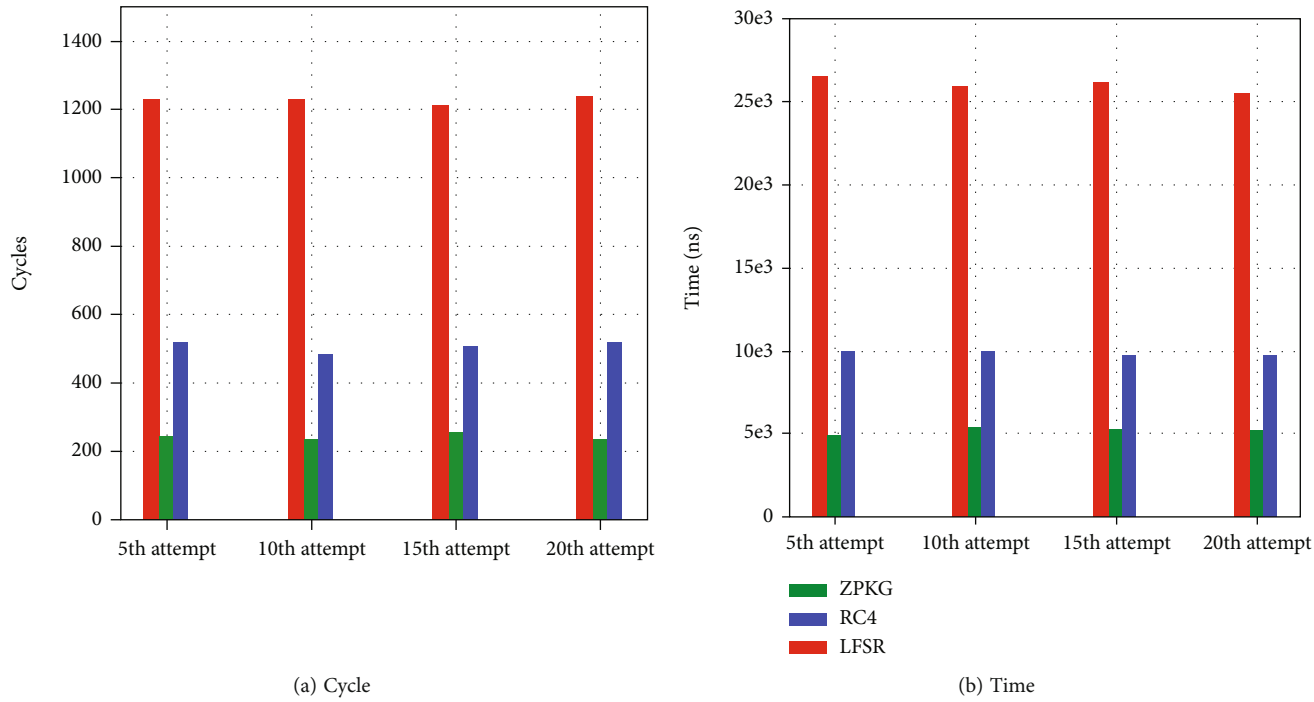
	ZPKG	RC4	LFSR
On-chip power	0.127 W	0.144 W	0.062 W
TJ *	26.5 °C	26.7 °C	26.5 °C
Thermal margin	58.5 °C	58.3 °C	26.2 °C
Off-chip power	0 W	0 W	0 W

*TJ: junction temperature.

Table 5: The key generation time.

	ZPKG	RC4	LFSR
Cycles	233	490	1275
Time (ns)	4670	9790	25500

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



(a) Cycle (b) Time

Figure 4: Statistics of key generation time and clock cycle of repeated attempts.

(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, α is chosen in the range $[0.001,0.01]$. Common values of α in cryptography are about 0.01 based on the NIST-800-22 test standard.

We configured four sets of initial parameters of N_0, M_0, e_1, and e_2.The experimental results of the NIST-800-22 test are shown in Table 2. Table 2 shows that the generated sequences of four sets of parameters passed all the tests. These sequences show good randomness and meet the requirements of the stream cipher.

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.

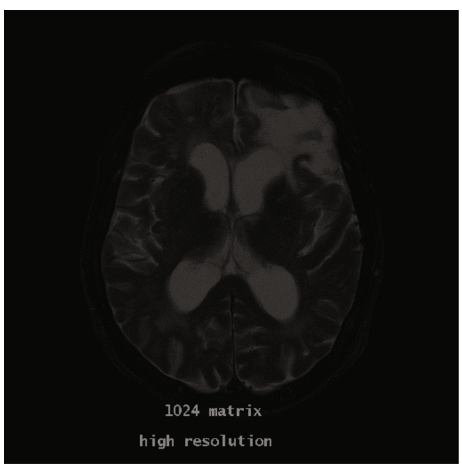
In this section, the ZPKG algorithm, RC4 algorithm, and LFSR algorithm are successfully applied to the encryption of more than 50 images of CVG-UGR test image set, including gray image, biometric image, medical image, and magnetic resonance image (MRI). The experimental simulation platform is FPGA, and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

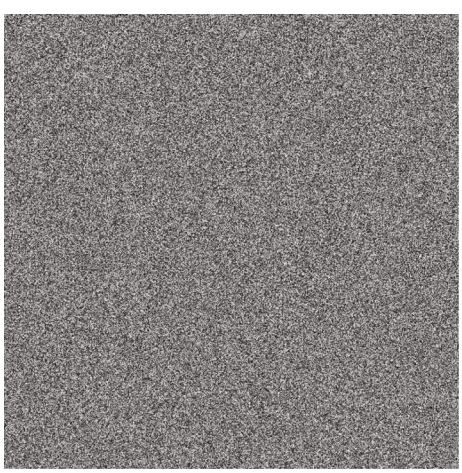
(a)

(b)



(c)

(d)



(e)

(f)

Figure 5: Continued.

(g) (h)

(i) (j)

(k) (l)

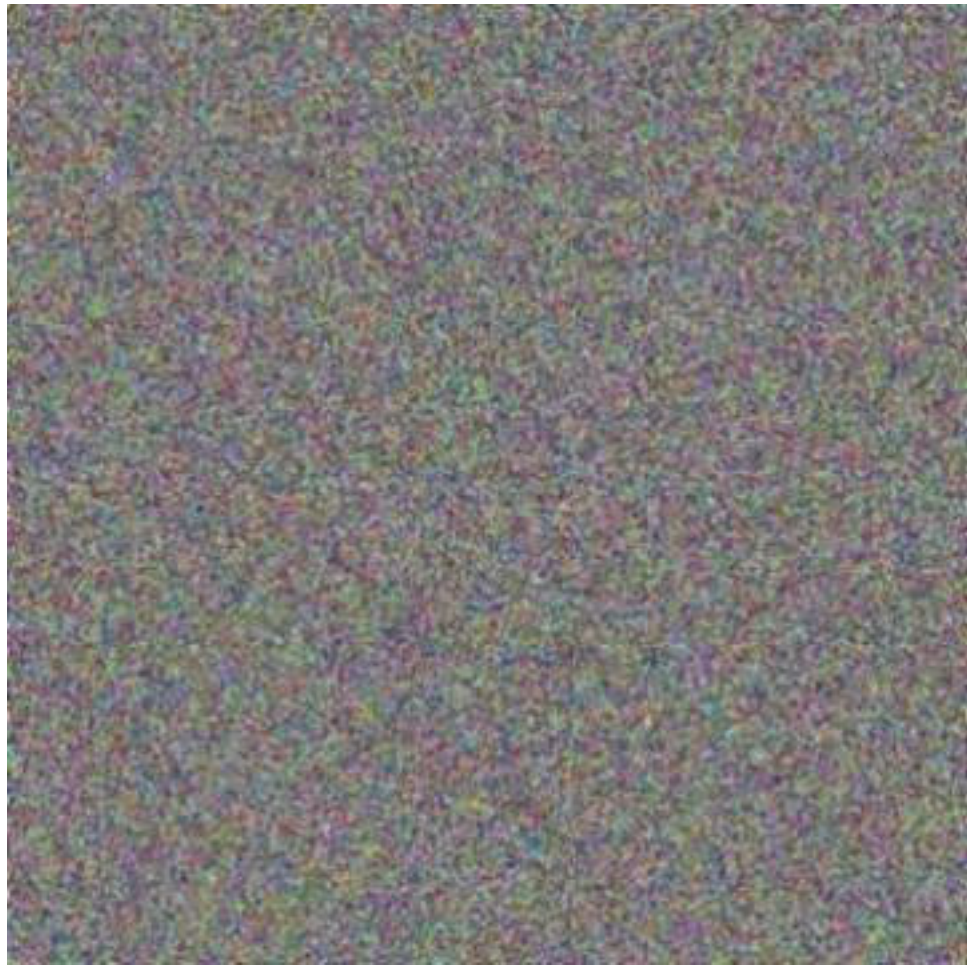
Figure 5: Grayscale images encryption and decryption using ZPKG algorithm. (a–d) Original. (e–h) Encrypted images. (i–l) Decrypted images (SSIM = 1).

(a)

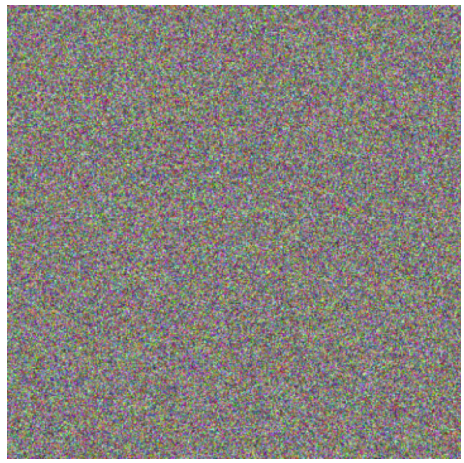
(b)

(c)

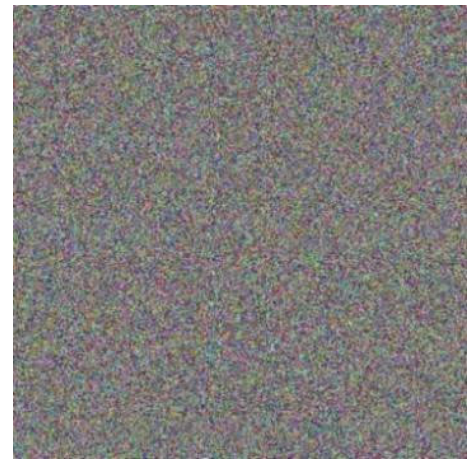
Figure 6: Continued.



(d)



(e)



(f)

Figure 6: Color image encryption using ZPKG algorithm. (a–c) Original. (d–f) Encrypted images.

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

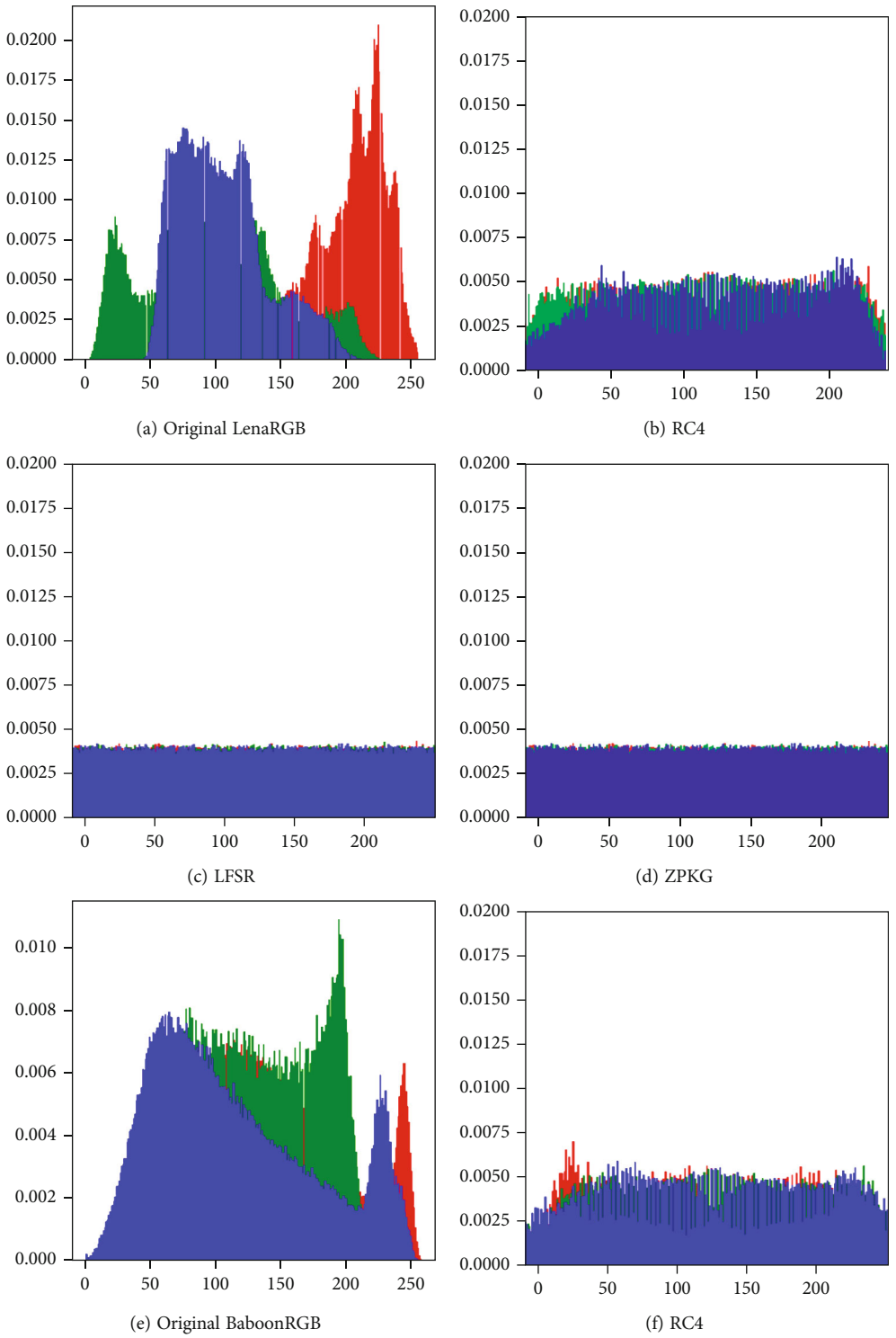


Figure 7: Continued.

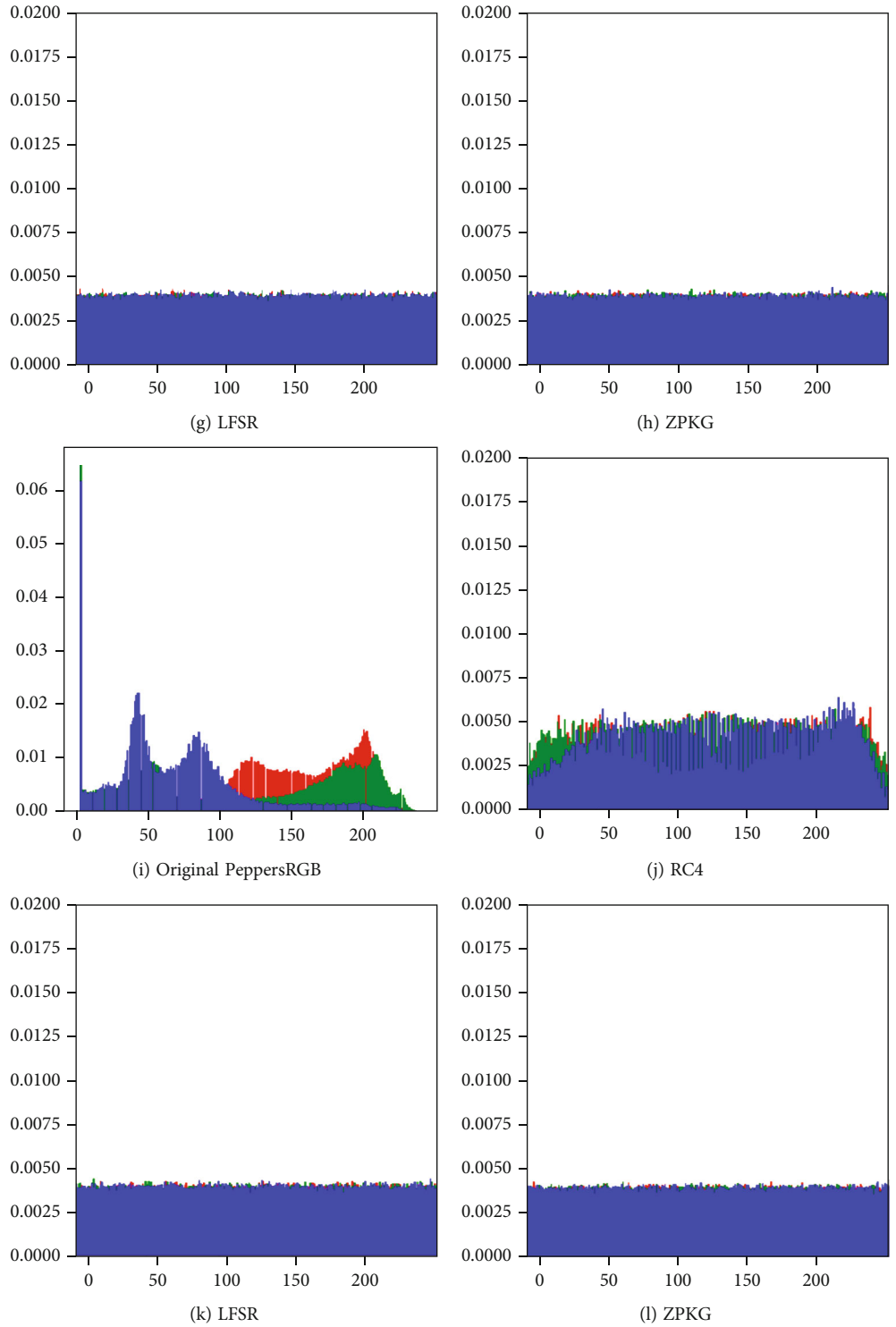
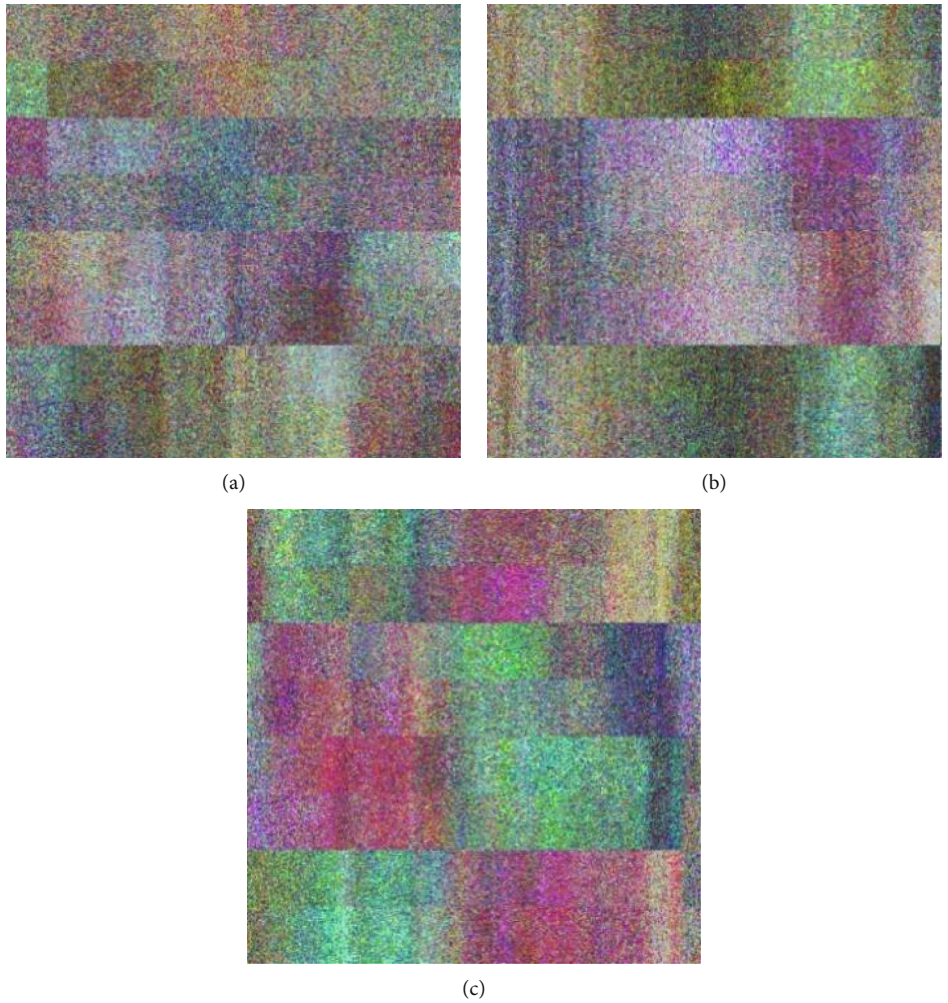


Figure 7: Color image histograms before and after encrypting with ZPKG, RC4, and LFSR.

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



(a) (b) (c)

Figure 8: Color image decryption using ZPKG with 1-bit flipped key. (a) Decrypt LenaRGB. (b) Decrypt BaboonRGB. (c) Decrypt PeppersRGB.

the SSIM index indicates that two measured images are identical. SSIM is computed as

$$\text{SSIM} = \frac{\text{Cov}(X, Y)}{\sqrt{\bar{X}\bar{Y}}}, \tag{34}$$

where

$$\begin{aligned} \text{Cov}(X, Y) &= N \sum_{i=1}^{N} x_i, y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i, \\ \bar{X} &= N \sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i \right)^2, \\ Y &= N \sum_{i=1}^{N} y_i^2 - \left(\sum_{i=1}^{N} y_i \right)^2, \end{aligned} \tag{35}$$

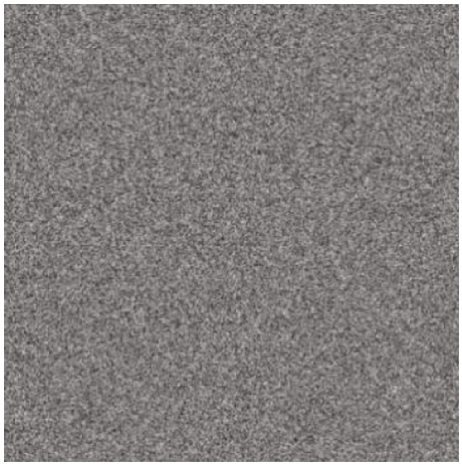
where x_i represents the gray value of the ith pixel of the first image, and y_i represents the gray value of the ith pixel of the second image.

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.

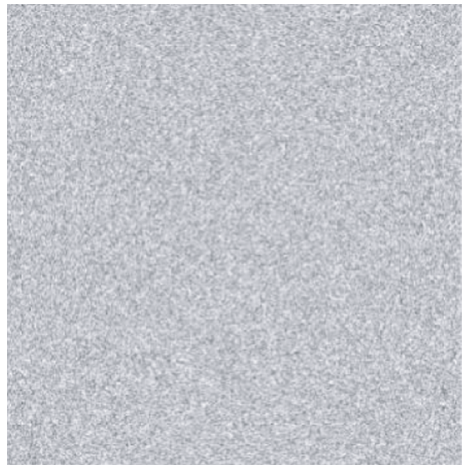
The 3D images such as color images contain several 2D data matrices called 2D components. Color images, for example, contain three color channels called R, G, and B. Each color channel is a 2D component. In this manner, the 3D images can be considered the combination of several 2D images. The 3D image 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.

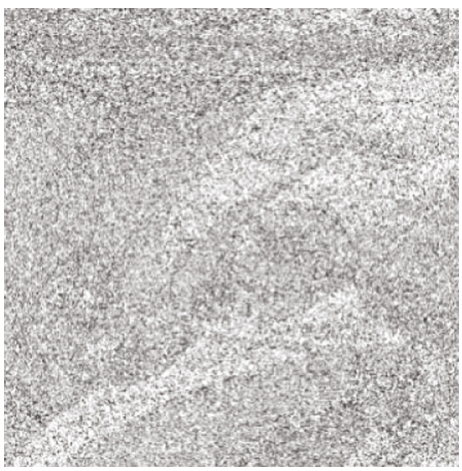
The encrypted image histograms using the ZPKG algorithm, LFSR algorithm, and RC4 algorithm are completely different from the original image (see Figure 7). Nevertheless,



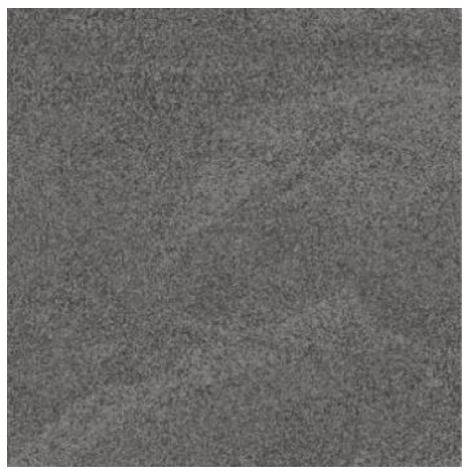
(a) ZPKG encrypt



(b) LFSR encrypt



(c) RC4-KDA encrypt

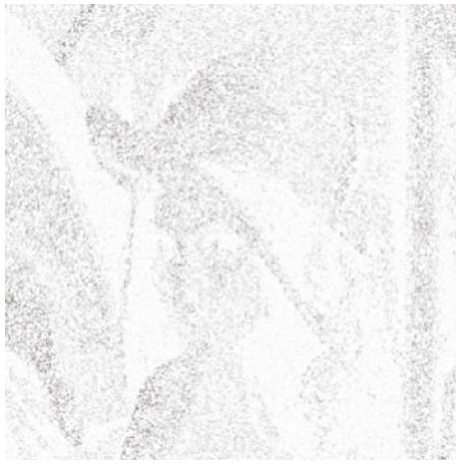


(d) RC4-XOR encrypt

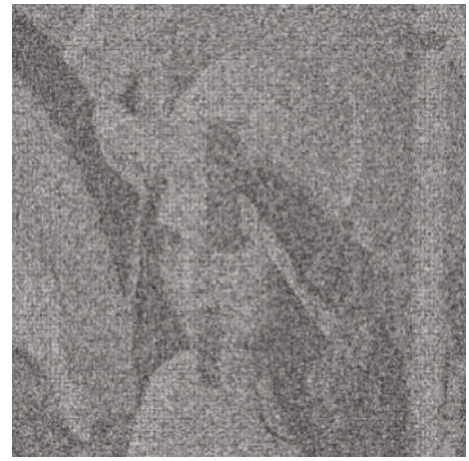
(e) ZPKG decrypt, SSIM = 0.9021

(f) LFSR decrypt, SSIM = 0.5068

Figure 9: Continued.



(g) RC4-KDA decrypt, SSIM = 0.7037



(h) RC4-XOR decrypt, SSIM = 0.4548

Figure 9: Encrypting and decrypting Lena image with noise overlap added.

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

[1] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[2] S. M. Seyedzadeh, B. Norouzi, and S. Mirzakuchaki, "RGB color image encryption based on Choquet fuzzy integral," *Journal of Systems and Software*, vol. 97, pp. 128–139, 2014.

[3] X. Wu, D. Wang, J. Kurths, and H. Kan, "A novel lossless color image encryption scheme using 2D DWT and 6D hyperchaotic system," *Information Sciences*, vol. 349-350, pp. 137–153, 2016.

[4] S. W. Golomb, *Shift Register Sequences*, Holden-Day Inc., San Francisco, 1967.

[5] E. Zeckendorf, "A generalized Fibonacci numeration," *Fibonacci Quarterly*, vol. 10, no. 4, pp. 365–372, 1972.

[6] C. G. Lekkerkerker, "Voorstelling van natuurlijke getallen door een som van getallen van fibonacci," in *Stichting Mathematisch Centrum*, pp. 190–195, Zuivere Wiskunde, 1951.

[7] M. Deza, "On minimal numbers of terms in representation of natural numbers as a sum of Fibonacci numbers," *Fibonacci Quarterly*, vol. 15, no. 3, pp. 237-238, 1977.

[8] M. Edson and L. Q. Zamboni, "On representations of positive integers in the Fibonacci base," *Theoretical Computer Science*, vol. 326, no. 1-3, pp. 241–260, 2004.

[9] E. P. Davlet'yarova, A. A. Zhukova, and A. V. Shutov, "Geometrization of the Fibonacci numeration system, with applications to number theory," *St Petersburg Mathematical Journal*, vol. 25, no. 6, pp. 893–907, 2014.

[10] N. K. Sreelaja and G. A. Vijayalakshmi Pai, "Stream cipher for binary image encryption using ant colony optimization based key generation," *Applied Soft Computing*, vol. 12, no. 9, pp. 2879–2895, 2012.

[11] "RC4 encryption algorithm," http://www.vocal.com.

[12] A. S. Mantin, "Weaknesses in the key scheduling algorithm of RC4," in *Selected Areas in Cryptography. SAC 2001*, S. Vaudenay and A. M. Youssef, Eds., vol. 2259 of Lecture Notes in Computer Science, pp. 1–24, Springer, Berlin, Heidelberg, 2001.

[13] Chung-Ping Wu and C. C. J. Kuo, "Design of integrated multimedia compression and encryption systems," *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 828–839, 2005.

[14] K. Khan, "Chaotic cryptography and its applications in telecommunication systems," *Telecommunication Systems*, vol. 52, no. 2, pp. 513-514, 2013.

[15] J. Leroy, M. Rigo, and M. Stipulanti, "Counting the number of non-zero coefficients in rows of generalized Pascal triangles," *Discrete Mathematics*, vol. 340, no. 5, pp. 862–881, 2017.

[16] C. Epifanio, C. Frougny, A. Gabriele, F. Mignosi, and J. Shallit, "Sturmian graphs and integer representations over numeration systems," *Discrete Applied Mathematics*, vol. 160, no. 4-5, pp. 536–547, 2012.

[17] J. Bernat, "Continued fractions and numeration in the Fibonacci base," *Discrete Mathematics*, vol. 306, no. 22, pp. 2828–2850, 2006.

[18] T. P. Berger, M. Minier, and B. Pousse, "Software oriented stream ciphers based upon FCSRs in diversified mode," in *Progress in Cryptology - INDOCRYPT 2009. INDOCRYPT 2009*, B. Roy and N. Sendrier, Eds., vol. 5922 of Lecture Notes in Computer Science, pp. 119–135, Springer, Berlin, Heidelber, 2009.

[19] Z. Lin, "The transformation from the Galois NLFSR to the Fibonacci configuration," in *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, pp. 335–339, Xi'an, China, 2013.

[20] S. S. Mansouri and E. Dubrova, "An improved hardware implementation of the Quark hash function," in *Radio Frequency Identification. RFIDSec 2013*, M. Hutter and J. M. Schmidt, Eds., vol. 8262 of Lecture Notes in Computer Science, pp. 113–127, Springer, Berlin, Heidelberg, 2013.

[21] U. Blocher and M. Dichtl, "Fish: a fast software stream cipher," in *Fast Software Encryption. FSE 1993*, R. Anderson, Ed., vol. 809 of Lecture Notes in Computer Science, pp. 41–44, Springer, Berlin, Heidelberg, 1993.

[22] G. T. Reddy, M. P. K. Reddy, K. Lakshmanna et al., "Analysis of dimensionality reduction techniques on big data," *IEEE Access*, vol. 99, pp. 54776–54788, 2020.

[23] N. Deepa, Q. V. Pham, D. C. Nguyen et al., "A Survey on Blockchain for Big Data: Approaches, Opportunities, and Future Directions," 2020, https://arxiv.org/abs/2009.00858.

[24] P. Filipponi and W. Wolfowicz, "A statistical property of non-adjacent ones binary sequences," *Note Recensioni Notizie*, vol. 36, no. 3, pp. 103–106, 1987.

[25] T. Mansour, "Combinatorial identities and inverse binomial coefficients," *Advances in Applied Mathematics*, vol. 28, no. 2, pp. 196–202, 2002.

[26] A. Doğanaksoy, F. Sulak, M. Uğuz, O. Şeker, and Z. Akcengiz, "New statistical randomness tests based on length of runs," *Mathematical Problems in Engineering*, vol. 2015, 14 pages, 2015.

[27] A. Rukhin, J. Soto, J. Nechvatal et al., *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, 2010.

[28] A. Popov, *RFC 7465: Prohibiting RC4 Cipher Suites*, IETF, 2015, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.682.1589&rep=rep1&type=pdf.

[29] S. L. Miao, J. Y. Zuo, and Y. F. Song, "Design and achieve of FPGA-based RC4 encryption algorithm," *Computer Measurement and Control*, vol. 26, no. 2, 2018.