A Fast Pseudorandom Sequence Generation Method Based on Sponge Functions (Postprint)

Authors: Zhao Lei, Zheng Dong, Ren Fang

Date: 2018-11-29T00:00:00+00:00

Abstract

Sponge functions utilize short keys and initialization vectors, and are thus employed as a novel structure for pseudo-random generators. Addressing issues such as low efficiency and slow speed in the 2SC (sponge code-based stream cipher) pseudo-random sequence generation method proposed by Meziani et al., we propose a fast pseudo-random sequence generation method based on sponge functions by incorporating coding theory. Using a universal state transformation, its security is reduced to the regular syndrome decoding problem, yet its computational capability is superior compared to regular coding. Theoretical analysis and experimental results demonstrate that this pseudo-random sequence generator retains the characteristics of sponge functions while achieving substantially improved efficiency. For a 160-bit security level, its speed is increased by more than fivefold compared to the original scheme. Furthermore, NIST statistical tests and evaluations of sequence balance, cross-correlation, and other properties indicate that the generated pseudo-random sequences exhibit excellent randomness characteristics.

Full Text

Preamble

Vol. 37 No. 1
Application Research of Computers
ChinaXiv Cooperative Journal

A Fast Pseudo-Random Sequence Generation Method Based on Sponge Functions

Zhao Lei^{a,b}, Zheng Dong^{a,b}, Ren Fang^{a,b†}

(a. School of Telecommunication & Information Engineering; b. National Engineering Laboratory for Wireless Security, Xi' an University of Posts & Telecommunications, Xi' an 710121, China)

Abstract: Sponge functions utilize relatively short keys and initialization vectors, making them a novel structure for pseudo-random generators. To address the low efficiency and slow speed of the 2SC (sponge code-based stream cipher) pseudo-random sequence generation method proposed by Meziani et al., this paper proposes a fast pseudo-random sequence generation method based on sponge functions combined with coding theory. Using a generic state transformation, its security is reduced to the Regular Syndrome Decoding (RSD) problem, yet it offers better computational characteristics than regular encoding. Theoretical analysis and experimental results demonstrate that this pseudo-random sequence generator retains the properties of sponge functions while significantly improving efficiency. For a 160-bit security level, its speed is more than 5 times faster than the original scheme. Meanwhile, NIST statistical tests and evaluations of sequence balance and cross-correlation indicate that the generated pseudo-random sequences exhibit excellent randomness properties.

Keywords: sponge function; pseudo-random sequence; coding theory; regular word; syndrome decoding

---

0 Introduction

Cryptography forms the foundation of network security. In today' s world where network activities have permeated every aspect of society—whether online banking, e-commerce, email, or instant messaging services—cryptographic technologies continuously protect user information security. Stream ciphers, as a mainstream encryption method, offer simple implementation, high speed, and relatively mature theory, making them widely applicable in practice, particularly in wireless communication standards such as IEEE 802.11b and Bluetooth. Pseudo-random sequences often serve as keystreams in stream ciphers, which are combined with plaintext streams through simple XOR operations to produce ciphertext streams. This "one-time pad" approach achieves perfect secrecy, the highest level of security. Consequently, keystream generation becomes the critical component of stream cipher algorithms. Currently, most keystream generators are based on pseudo-random number generators (PRNGs), making the design of high-performance PRNGs a research hotspot in cryptography.

When applied in stream ciphers, pseudo-random sequences require not only favorable statistical properties and efficiency but also provable security—meaning their security must be reducible to specific hard problems. Among existing pseudo-random sequence generation methods, those based on Linear Feedback Shift Registers (LFSR) are common, yet they suffer from numerous security weaknesses, particularly low linear complexity. To resist attacks, Blum et al. proposed in the early 1980s a provably secure pseudo-random sequence generator

based on the integer factorization problem. Subsequently, Kaliski introduced another scheme whose security relies on the difficulty of the discrete logarithm problem, while Alexi et al. proposed a provably secure method based on the RSA problem.

Although the aforementioned pseudo-random sequence generation methods are provably secure, they would lose their security under quantum attacks. Therefore, designing pseudo-random number generators based on alternative assumptions is crucial, representing a direction in post-quantum cryptography research. The earliest post-quantum resistant pseudo-random sequence generator was proposed by Impagliazzo et al., introducing a provably secure method based on the subset sum problem. Later, Fischer and Stern proposed a pseudo-random sequence generator based on the Syndrome Decoding (SD) problem, which suffered from two defects: slow computation speed and large memory consumption for syndrome calculation matrices, making it difficult to apply in practice. In 2007, Gaborit et al. introduced a new scheme called SYND, an improvement over the previous work, with its security reduced to the SD problem. In 2011, Meziani et al. proposed a new pseudo-random sequence generation method based on the sponge structure and syndrome decoding problem, called 2SC. This cryptographic structure proved more efficient than SYND in performance metrics, achieving high security levels with small key sizes and initialization vectors. However, its primary drawback was the requirement for large matrices. In 2016, Gaborit et al. constructed a pseudo-random sequence generator based on the difficulty of the Rank Metric code syndrome decoding problem.

This paper re-examines the 2SC scheme proposed by Meziani et al. and presents an efficient improved algorithm. While retaining the sponge structure to keep keys and initialization vectors short, our approach abandons the process of encoding codewords into regular words and instead employs the construction philosophy of compression functions in hash algorithms, transforming it into an encoding method reducible to the regular word syndrome decoding problem. This results in significantly faster pseudo-random sequence generation.

---

1 Coding Theory

This section reviews the fundamentals of code-based cryptography, briefly describes common hard problems in coding theory, and finally introduces hash functions based on coding.

1.1 Linear Block Codes

A linear block code $\mathcal{C}$ of length n and dimension k over finite field $\mathbb{F}_q$ is defined as a k-dimensional linear subspace of the n-dimensional linear space $\mathbb{F}_q^n$. Vectors in $\mathbb{F}_q^n$ are called words, while vectors in $\mathcal{C}$ are called codewords. Here, n is called the block length and k the dimension of the block code.

Definition 1 A generator matrix of an $[n, k]$ linear block code $\mathcal{C}$ is a $k \times n$ matrix G whose row vectors form a basis of $\mathcal{C}$. The generator matrix of a linear block code is not unique; different generator matrices can be converted through elementary row operations. That is, if G is a generator matrix of $\mathcal{C}$ and P is an elementary matrix, then PG is also a generator matrix of $\mathcal{C}$.

Definition 2 A parity-check matrix of an $[n, k]$ linear block code $\mathcal{C}$ is an $(n - k) \times n$ matrix H where any row vector of H is orthogonal to any row vector of the generator matrix G of $\mathcal{C}$, i.e., $HG^T = 0$. The parity-check matrix of a linear block code is not unique and can be converted through elementary row operations. A vector c of length n is a codeword of $\mathcal{C}$ if and only if $Hc^T = 0$. For any word x, Hx^T is called the syndrome of x.

Definition 3 For two codewords $u = (u_1, u_2, \dots, u_n)$ and $v = (v_1, v_2, \dots, v_n)$ of a linear block code $\mathcal{C}$, the Hamming distance $d(u, v)$ is defined as the number of different components between u and v, i.e., $d(u, v) = |\{i|u_i \neq v_i\}|$. The Hamming weight $w(u)$ of a codeword u is defined as the number of non-zero components of u. The minimum Hamming weight among all non-zero codewords of $\mathcal{C}$ is called the minimum distance of $\mathcal{C}$, denoted as $d(\mathcal{C})$. The minimum distance determines the error-correcting capability of the code. Generally, the error-correcting capability t of a linear block code satisfies $t = \lfloor (d - 1)/2 \rfloor$.

Definition 4 Given a word of length n and weight w, convert it into w blocks of length n/w, each containing exactly one "1" . Such a codeword is called a regular word.

1.2 Hard Problems

In code-based cryptography, security is mostly reduced to the following hard problems:

Q1 Syndrome Decoding Problem (SD)
Given an $(n - k) \times n$ matrix H over finite field $\mathbb{F}_q$ and a vector $s \in \mathbb{F}_q^{n-k}$, does there exist a word $x \in \mathbb{F}_q^n$ with weight less than or equal to w such that $Hx^T = s$?

Q2 Codeword Finding Problem (CF)
Given an $(n-k) \times n$ matrix H over $\mathbb{F}_q$ and an integer $w > 0$, does there exist a non-zero word $x \in \mathbb{F}_q^n$ with weight less than or equal to w such that $Hx^T = 0$?

The SD and CF problems have been proven to be NP-complete in the literature [14].

A special case of the SD problem is the RSD problem, which was proven to be NP-complete in [13] and is described as follows:

Q3 Regular Syndrome Decoding Problem (RSD)
Given an $(n - k) \times n$ matrix H over finite field $\mathbb{F}_q$ and a vector $s \in \mathbb{F}_q^{n-k}$, do there exist w vectors $h_{i_1}, h_{i_2}, \dots, h_{i_w}$, which are column vectors of H, such that s equals the sum of these vectors?

1.3 Construction of Hash Functions Based on Coding

The core component in hash function construction is the compression function F. The construction method for compression functions based on coding hard problems is as follows:

Randomly select an $[n, k]$ codeword C, where: - $m = n/2$ - $t = n - k - r$ - $w = |m|_2$

The length n input word x can be divided into w equal-length blocks, each containing m bits.

Randomly select an $(n - k) \times n$ parity-check matrix H, which can be divided into w submatrices, i.e., $H = (H_1, H_2, \ldots, H_w)$, where each submatrix H_i is the i-th column of matrix H.

Define the compression function $g : \mathbb{F}_2^m \to \mathbb{F}_2^{n-k}$. For any $x \in \mathbb{F}_2^m$, also divide x into w blocks using the above method: $x = (x_1, x_2, \ldots, x_w)$. Convert each x_i into a number between 0 and $2^m - 1$, select the x_i-th column of submatrix H_i, then the function output is:

$$g(x) = \sum_{i=1}^{w} h_{x_i}$$

Theorem 1 The output of the above compression function g is equivalent to computing the syndrome of a regular word of length n and weight w, i.e., for any $x \in \mathbb{F}_2^m$, there exists a regular word c such that $g(x) = Hc^T$. Proof can be found in reference [15].

---

2 Sponge Function and 2SC Scheme

2.1 Sponge Function

The sponge function was first proposed by Peeters et al. in [16], providing a new approach for iterative hash function design. It uses a finite state to receive input bitstreams of any length and can produce output of any length.

The sponge function consists of three parts, as shown in Figure 1 [Figure 1: see original paper]: a) A memory state S containing b bits. The memory state is divided into two blocks: R (size r bits) and C (size c bits). The parameter r is called the bitrate, while c is called the capacity. b) A fixed-size transformation function f that permutes or transforms the memory state. c) A padding function *pad* that adds sufficient length to the input to make its bitstream length an integer multiple of r.

Thus, the padded input can be split into an integer number of segments of length r. The specific operation of the sponge function is as follows: a) Initialize S to zero. b) Process the input through the padding function. c) XOR the first r bits of the padded input with R. d) Transform S through function f to obtain

S'. e) If there is remaining padded input, XOR the next r bits with R' of S'. f) Transform S' through function f to obtain S''.

This process repeats until all input is consumed, called the absorbing phase (in the sponge metaphor, this represents being "absorbed" by the function).

The output block Z_i quantity is user-selected, and this phase is called the squeezing phase ("squeezing out"), with the specific process as follows: a) After the absorbing phase, obtain the new memory state S. At this point, Z_0 represents the first r bits of output. b) If more output is needed, transform S through f to obtain S'. c) At this point, Z_1 represents the next r bits of output.

This process repeats until the required output length is satisfied. Note that input is never XORed with the C portion of memory, and this portion is never directly output. This memory portion is only related to the transformation function. In hash functions, preventing collision attacks or preimage attacks relies on this memory portion.

2.2 2SC Scheme Structure

Based on the sponge function described above, Meziani et al. designed a pseudo-random sequence generation method. Let K and IV represent the key and initialization vector respectively, with $|K| = |IV| = k$. The internal state S has size b bits.

Initialization The initialization function takes key K and initialization vector IV as input, computes and returns the initial state S_0, defined as follows:

$$S_0 = f(K|IV|0^{c-r})$$

where "|" denotes concatenation and 0^{c-r} is a zero vector of size $c-r$ bits. $[r]_1^f$ represents the first r bits of f, and $[c]_1^f$ represents the remaining c bits. The syndrome mapping is defined as:

$$x \xrightarrow{\phi} g(x) = H\phi(x)^T$$

Here, function ϕ denotes regular encoding, converting an r-bit string into a regular word of length n and weight w. Matrix H is an $n \times s$ random matrix. This initialization step requires computing two functions and performing two XOR operations, making it more efficient than the SYND scheme when optimal parameters are selected for each security level. Through this initialization process, the initial state S_0 is obtained.

Update In this step, a new random function f is used to update the internal state (e.g., N times). The number of runs is user-selected, affecting both the security and efficiency of the structure. Function f is defined as:

$$S_{i+1} = f(S_i)$$

where:

$$f(x) = \begin{cases} x \oplus a_1 & \text{if } x \in \mathbb{F}_2^r \\ x \oplus b_1 & \text{if } x \in \mathbb{F}_2^c \end{cases}$$

A and B are $b \times b$ binary random matrices, A_i and B_i are submatrices of A and B respectively, $a_{i,j}$ represents the j-th column of the i-th submatrix of A, and $b_{i,j}$ represents the j-th column of the i-th submatrix of B.

Squeezing After obtaining S_i through the update process, 2SC finally produces keystream blocks of size r bits (Z_i), described as follows:

$$Z_i = [r]_1^{f(S_i)}$$

where $[r]_1^{f(S_i)}$ represents the first r bits of the output after applying the update function to the internal state S_i.

The structure described above is shown in Figure 2 [Figure 2: see original paper].

---

3 A Fast Pseudo-Random Sequence Generation Method Based on Sponge Functions

During experimental simulation, we found that the pseudo-random sequence generation method introduced in the previous section consumes significant time in the regular encoding Niederreiter transformation [17] process. Therefore, we aim to improve its performance through an alternative encoding approach without reducing security. Based on the construction methodology of hash functions from coding theory, we improve the above scheme as specifically described below.

Given input x, composed of blocks $x_1, x_2, \ldots, x_w$, with $|x| = m$ bits, this paper first provides data for each block through function f_1 to obtain output y_i, then performs XOR operations on y_i to obtain the final output. The structure is shown in Figure 3 [Figure 3: see original paper].

The function is specifically described as: Select a binary random matrix H of size $n \times s$, divide it into w submatrices of size $(n-k) \times s/w$:

$$H = (H_1, H_2, \ldots, H_w)$$

For any $x \in \mathbb{F}_2^m$, also divide x into w blocks using the above method: $x = (x_1, x_2, \ldots, x_w)$. Convert each x_i to a number between 0 and $2^b - 1$, select the x_i-th column of submatrix H_i, then the function output is:

$$D(x) = \bigoplus_{i=1}^{w} h_{x_i}$$

where each submatrix H_i is $(n-k)\times(n/w)$, and each submatrix has 2^b columns. Let j equal the decimal value of x_i, then $h_{i,j}$ is obtained through function f_1, with each x_i having 2^b possibilities. The functions f_1 and g are redefined as follows:

$$f_1(x) = a_1(x_1) \oplus a_2(x_2) \oplus \cdots \oplus a_w(x_w)$$
$$g(x) = b_1(x_1) \oplus b_2(x_2) \oplus \cdots \oplus b_w(x_w)$$

where A and B are $b \times b$ binary random matrices, A_i and B_i are submatrices of A and B respectively, $a_{i,j}$ represents the j-th column of the i-th submatrix of A, and $b_{i,j}$ represents the j-th column of the i-th submatrix of B.

The following example illustrates the above method. First, let $w = 3$ and $b = 2$. Matrix A is a 6×6 matrix. Randomly select a matrix A that meets the above requirements as follows:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Let $x = (00|01|10)$, divide x into groups and represent them in decimal: $x'_1 = 0$, $x'_2 = 1$, $x'_3 = 2$. According to the formula $z_i = (2^{b-1} + x_i) \cdot 2^{b(i-1)}$, we calculate $z_1 = 1$, $z_2 = 6$, $z_3 = 9$. In binary notation, $z = (001|010|001|000|010|000)$. Therefore, we can verify:

$$D(x) = A \cdot z^T = a_{0,3} \oplus a_{1,2} \oplus a_{2,1} = (1111)^T$$

---

4 Security Analysis

This chapter analyzes the algorithm' s security from two perspectives. Theoretically, it is difficult to reverse-engineer the key K and initialization vector IV. In practical attacks, adversaries must solve the SD problem. Currently, there are two effective attacks against the SD problem: Information Set Decoding (ISD) and Generalized Birthday Attack (GBA).

4.1 Theoretical Security

From the transformation function $D(x)$ described in Section 3, define a generic function:

$$D(x) = g(f_1(x))$$

where $f_1 : \mathbb{F}_2^r \to \mathbb{F}_2^n$ and $g : \mathbb{F}_2^n \to \mathbb{F}_2^{n-k}$. For any $x \in \mathbb{F}_2^r$, we need to prove two points:

a) For each input codeword x, there exists a regular word z such that $D(x) = H \cdot z^T$. Solving $D(x)$ is equivalent to finding a regular word z with weight less than or equal to w such that $Hz^T = D(x)$. This is precisely an instance of RSD. Therefore, the security of the improved scheme can be reduced to the RSD problem.

b) It has been proven that for each input x, a regular word z of weight w can be found such that $D(x) = H \cdot z^T$. Suppose there exists an adversary that can reverse-engineer $D(x)$, i.e., given $D(x)$, the adversary outputs x. Then given a matrix H and a value s, the same adversary could output a regular word z such that $Hz^T = s$. This is precisely an instance of RSD. Therefore, the algorithm' s security can be reduced to the hardness of the RSD problem, which, like the SD problem, is NP-complete.

Proof of a)
First, $H = (H_1, H_2, \dots, H_w)$, where each submatrix H_i has size $(n-k) \times (n/w)$. Each submatrix has 2^b columns selected from $\{0,1\}^b$. For a regular word z of length n and weight w, let $z_1, z_2, \dots, z_w$ represent the positions of its non-zero entries (since the word is regular, each value between z_i and z_{i+1} is unique). Then z and x have the relationship:

$$z_i = (2^{b-1} + x_i) \cdot 2^{b(i-1)}$$

The inverse transformation from z to x is:

$$\begin{cases} x_1 \equiv z_1 - 2^{b-1} \pmod{2^b} \\ x_2 \equiv (z_2 - z_1) - 2^{b-1} \pmod{2^b} \\ \vdots \\ x_w \equiv (z_w - z_{w-1}) - 2^{b-1} \pmod{2^b} \end{cases}$$

Therefore, it is easy to verify that $D(x) = H \cdot z^T$.

4.2 Practical Security

In practice, adversaries face two problems. On one hand, they obtain r bits of output but cannot access the remaining c bits; the larger the capacity, the more secure the system. On the other hand, even if they successfully guess these bits, they must solve an instance of the RSD problem. For appropriately chosen parameters, effectively solving the RSD problem is as difficult as solving the SD problem. Indeed, all known SD attacks are fully exponential; only two algorithms can attack systems based on the SD problem: Information Set Decoding (ISD) [10] and Generalized Birthday Attack (GBA) [18]. The latest GBA algorithm against code-based cryptosystems was proposed in [19] and will be used to select security parameters for 2SC.

Note that another attack called Memory Trade-off Attacks exists, initially introduced in [20] as a generic method for attacking block ciphers. To avoid such

attacks, the initialization vector IV should be at least as large as the key K, and c should be at least twice the key size.

Based on the descriptions in this and previous sections, we compare the 2SC scheme with our proposed scheme structurally, as shown in Table 1 . The next chapter will present specific experimental comparisons.

Table 1 Structural Comparison Between 2SC Scheme and Proposed Scheme

Scheme	Cycle Structure	Dependent Hard Problem	Problem Nature	Characteristics
2SC	Sponge Structure	Niederreiter Transform	SD Problem	NP-Complete, Time-consuming
Proposed	Sponge Structure	Coding-based Hash Transform	RSD Problem	NP-Complete, Less Time-consuming

5 Experimental Simulation

We analyze the improved pseudo-random sequence generation algorithm, compare it with the original 2SC algorithm, and test the randomness of the generated sequences.

5.1 Algorithm Efficiency Analysis

Parameter selection must achieve high efficiency and security under known attacks. First, according to Time Memory Trade-Off attacks, c should be at least twice the key size. Select the optimal value $b = \lfloor 8/\log_2 w \rfloor$. For each security level λ, solving the RSD problem has complexity at least 2^λ. We tested the time required to generate $N = 10^6$ bits of pseudo-random sequence under different security levels. Table 2 presents test results for several security levels by selecting optimal parameter sets. Note that during simulation implementation, only random binary matrices were used without any specific structure. However, when the parity-check matrix is quasi-cyclic, parameters providing the same security level can be found [10].

The results in Table 3 were implemented in Pycharm using Python on a Windows 10 operating system with a 2.3GHz CPU. For different security levels, 10^6 bits of data were generated in 10 iterations. As the parity-check matrix size increases, sequence generation time increases accordingly, but security also improves. Table 3 compares the implementation results of the 2SC algorithm with the new algorithm. From Table 2 , we can draw the following conclusions: For an 80-bit security level, generating 10^6 bits of pseudo-random sequence, our

method is 3.19 times faster than the original 2SC method; for a 120-bit security level, it is 3.02 times faster; for a 160-bit security level, it is 5.75 times faster; for a 400-bit security level, it is 4.5 times faster.

Table 2 Test Results Under Different Security Levels

Key Size (bits)	Matrix Size	w	b	Time (s)
80	1700×3400	20	8	0.43
120	2500×5000	25	8	1.44
160	3300×6600	33	8	3.06
400	8100×16200	81	8	31.2

Note: Test time does not include initialization time

Table 3 Performance Comparison Under Different Security Levels

Key Size (bits)	2SC (s)	Our Scheme (s)	Ratio
80	1.37	0.43	3.19
120	4.35	1.44	3.02
160	17.60	3.06	5.75
400	140.40	31.20	4.50

5.2 NIST SP800-22 Test Analysis

We conducted randomness tests on the algorithm' s generated pseudo-random sequences using the NIST SP800-22 test suite sts-2.1.1 [21]. The test suite contains 15 test items, requiring sequence length greater than 10^6 bits. When test value P-value ≥ 0.01, the sequence is considered random in that test; when P-value < 0.01, it is considered non-random. We used sts-2.1.1 to test 6×10^7 bits of pseudo-random sequence generated by the algorithm. The tests were divided into 6 groups, each of length 10^7 bits. The test results are shown in Table 4 .

The results show that among 15 test items, only one test value failed, with all others greater than the significance level $\alpha = 0.01$. In the Frequency Test, the mean value greater than 0.5 indicates that the numbers of 0s and 1s in the pseudo-random sequence are nearly equal, showing good balance. The Runs test mean of 0.375 indicates that the number of runs is close to that of a truly random sequence. The Serial test mean greater than 0.5 shows that occurrences of subsequences of specified lengths tend to be equiprobable. The Cumulative Sums test values all above 0.6 indicate uniform distribution of 0s and 1s. The Linear Complexity test values all above 0.8 indicate high sequence complexity. Therefore, the pseudo-random sequences generated by our method exhibit excellent randomness characteristics.

Table 4 Test Results of Randomness

Test Item	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
Frequency	0.534146	0.739918	0.224821	0.350485	0.534146	0.911413
Block Frequency	0.739918	0.739918	0.911413	0.739918	0.224821	0.534146
Cumulative Sums	0.911413	0.350485	0.534146	0.739918	0.534146	0.350485
Longest Run of Ones	0.534146	0.534146	0.739918	0.739918	0.739918	0.911413
Discrete Fourier Transform	0.224821	0.350485	0.534146	0.739918	0.739918	0.350485
Non-periodic Template	0.350485	0.534146	0.350485	0.534146	0.739918	0.534146
Overlapping Template	0.739918	0.534146	0.739918	0.350485	0.534146	0.739918
Universal Statistical	0.534146	0.739918	0.350485	0.739918	0.350485	0.534146
Approximate Entropy	0.739918	0.534146	0.739918	0.534146	0.739918	0.350485
Random Excursions	0.350485	0.534146	0.739918	0.739918	0.350485	0.534146
Random Excursions Variant	0.534146	0.739918	0.350485	0.534146	0.739918	0.350485
Serial	0.739918	0.534146	0.739918	0.350485	0.534146	0.739918
Linear Complexity	0.911413	0.739918	0.739918	0.911413	0.739918	0.739918

Note: Underlined test values indicate failure

5.3 Correlation Analysis

The autocorrelation and cross-correlation coefficients of sequences are commonly used statistics for evaluating randomness [22]. For a pseudo-random sequence $\{b_i\}$ of length N, the autocorrelation coefficient is defined as:

$$ac(m) = \frac{1}{N}\sum_{i=1}^{N-m}(2b_i - 1)(2b_{i+m} - 1)$$

where m is the step size. The autocorrelation coefficient depends on the step size; smaller variation when the step size changes indicates better randomness performance. The theoretical value is 0.5 when step size $m = 0$, and 0 for other step sizes.

The cross-correlation coefficient between sequences $\{b_i\}$ and $\{c_i\}$ is defined as:

$$cc(m) = \frac{1}{N} \sum_{i=1}^{N-m} (2b_i - 1)(2c_{i+m} - 1)$$

The theoretical value is 0. Values closer to theoretical values indicate better randomness. We randomly selected two groups of pseudo-random sequences of length $N = 38400$ for testing. The experimental results are shown in Figure 4 [Figure 4: see original paper].

The experimental results show that the distributions of both autocorrelation and cross-correlation are close to ideal, indicating low correlation between sequences generated by our method and good randomness.

5.4 Balance Analysis

For a pseudo-random sequence of length N, let the numbers of 0s and 1s be n_0 and n_1 respectively. The balance degree is defined as:

$$E(N) = \frac{|n_0 - n_1|}{N}$$

A smaller balance degree value indicates that the numbers of 0s and 1s are more similar, implying better randomness. We selected three groups of sequences of length $N = 38400$ (c_1, c_2, c_3) for testing. Figure 5 [Figure 5: see original paper] shows the experimental balance degree results.

The experimental results show that the balance degrees of all three pseudo-random sequences are below 0.010, close to 0. This indicates that the numbers of 0s and 1s in the sequences are very close, demonstrating stable performance and good balance.

5.5 Run Test

A run is a subsequence of a pseudo-random sequence consisting of consecutive 0s or 1s, where the elements before and after differ from the run elements. The number of consecutive occurrences is called the run length. In random sequences, the numbers of 0s and 1s of any length are approximately equal, where runs of length i constitute about $1/2^{i+1}$ of the total sequence length. The closer the generated pseudo-random sequence is to the ideal state, the better its randomness.

We conducted run statistical tests on three groups of sequences. The specific statistics are shown in Tables 5 through 7 .

Table 5 Run Statistics of Sequence c_1

Run Length	1	2	3	4	5	6	7	8
Measured	0.5012	0.2487	0.1260	0.0604	0.0333	0.0150	0.0076	0.0044
Theoretical	0.5000	0.2500	0.1250	0.0625	0.0313	0.0156	0.0078	0.0039

Table 6 Run Statistics of Sequence c_2

Run Length	1	2	3	4	5	6	7	8
Measured	0.5027	0.2527	0.1218	0.0600	0.0316	0.0154	0.0078	0.0044
Theoretical	0.5000	0.2500	0.1250	0.0625	0.0313	0.0156	0.0078	0.0039

Table 7 Run Statistics of Sequence c_3

Run Length	1	2	3	4	5	6	7	8
Measured	0.5020	0.2478	0.1233	0.0629	0.0333	0.0158	0.0078	0.0043
Theoretical	0.5000	0.2500	0.1250	0.0625	0.0313	0.0156	0.0078	0.0039

The experimental results from all three groups show that the run statistics of the generated pseudo-random sequences are close to theoretical values, with nearly equal numbers of 0s and 1s. The percentages of sequences with different run lengths are comparable to theoretical values. Therefore, the sequences obtained in our experiments conform to run statistical characteristics.

6 Conclusion

This paper proposes a fast pseudo-random sequence generation method based on sponge functions, representing an improvement over the 2SC pseudo-random sequence generation method. This approach shortens the key and initialization vector lengths while replacing the process of encoding codewords into regular words with an encoding method reducible to the regular word syndrome decoding problem, substantially increasing computational speed. Experimental simulation results demonstrate that our method achieves at least a 3x improvement in sequence generation speed compared to the original method at the same security level. NIST test results confirm that the pseudo-random sequences generated by our method exhibit excellent randomness, with autocorrelation, cross-correlation, balance degree, and run statistics all matching theoretical values. Moreover, the scheme' s security relies on the RSD problem, enabling resistance against existing quantum attacks.

References

[1] Zhang Bin, Xu Chao, Feng Dengguo. Design and analysis of stream ciphers: past, present and future directions [J]. Journal of Cryptologic Research, 2016, 3(6): 527-545.

[2] Avaroğlu E, Koyuncu İ, Özer A B, et al. Hybrid pseudo-random number generator for cryptographic systems [J]. Nonlinear Dynamics, 2015, 82(1-2): 58-61.

[3] Wang Yong, Liu Zhaolong, Ma Jianbin, et al. A pseudorandom number generator based on piecewise logistic map [J]. Nonlinear Dynamics, 2016, 83(4): 2373-2391.

[4] Blum M, Micali S. How to generate cryptographically strong sequences of pseudo random bits [C]// Proc of Symposium on Foundations of Computer Science. 2008: 112-117.

[5] Bernstein D J, Hamburg M, Krasnova A, et al. Elligator: elliptic-curve points indistinguishable from uniform random strings [C]// Proc of ACM SIGSAC Conference on Computer & Communications Security. New York: ACM Press, 2013: 967-980.

[6] Alexi W, Chor B, Goldreich O, et al. RSA and Rabin functions: certain parts are as hard as the whole [J]. SIAM Journal on Computing, 1988, 17(2): 194-209.

[7] Liu Wenrui. Quantum computing attack password system development analysis [J]. Communications Technology, 2017, 50(5): 1054-1059.

[8] Impagliazzo R, Naor M. Efficient cryptographic schemes provably as secure as subset sum [C]// Proc of Symposium on Foundations of Computer Science. 1989: 236-241.

[9] Fischer J B, Stern J. An efficient pseudo-random generator provably as secure as syndrome decoding [C]// Proc of International Conference on Theory and Application of Cryptographic Techniques. Springer-Verlag, 1996: 245-255.

[10] Gaborit P, Lauradoux C, Sendrier N. SYND: a fast code-based stream cipher with a security reduction [C]// Proc of IEEE International Symposium on Information Theory. IEEE Xplore, 2007: 186-190.

[11] Meziani M, Cayrel P L, Alaoui S M E Y. 2SC: an efficient code-based stream cipher [C]// Proc of International Conference on Information Security and Assurance. Berlin: Springer, 2011: 111-122.

[12] Gaborit P, Hauteville A, Tillich J P. RankSynd a PRNG based on rank metric [M]// Post-Quantum Cryptography. Springer International Publishing, 2016.

[13] Augot D, Finiasz M, Sendrier N. A Family of Fast Syndrome Based Cryptographic Hash Functions [C]// Proc of International Conference on Cryptology

in Malaysia. Berlin: Springer, 2005: 64-83.

[14] Berlekamp E R, Mceliece R J, Van Tilborg H C A. On the inherent intractability of certain coding problems (Corresp.) [J]. IEEE Trans. on Information Theory, 1978, 24(3): 384-386.

[15] Ren Fang, Zheng Dong. An improved code-based digital signature algorithm [J]. Journal of XI' AN University of Post and Telecommunications, 2015, 20(5): 38-43.

[16] Bertoni G, Daemen J, Peeters M, et al. Sponge functions [J]. Ecrypt Hash Workshop, 2007.

[17] Overbeck R, Sendrier N. Code-based cryptography [M]// Post-Quantum Cryptography. Springer Berlin Heidelberg, 2009: 95-145.

[18] Shenghui Su, Tao Xie, Shuwang Lü. A provably secure non-iterative hash function resisting birthday attack [J]. Theoretical Computer Science, 2016, 654(22): 128-142.

[19] Finiasz M, Sendrier N. Security Bounds for the Design of Code-Based Cryptosystems [C]// Proc of International Conference on Theory and Application of Cryptology and Information Security. Proceedings. 2009: 88-105.

[20] Hellman M. A cryptanalytic time-memory trade-off [J]. IEEE Trans on Information Theory, 1980, 26(4): 401-406.

[21] Rukhin A, Soto J, Nechvatal J, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications [J]. Applied Physics Letters, 2015, 22(7): 1645-179.

[22] Han Rui, Zhang Xuefeng. Pseudo-random sequence generating method based on high dimensional cat map [J]. Computer Engineering and Applications, 2016, 52(10): 91-99.

Note: Figure translations are in progress. See original paper for figures.