

# A New Construction of Pseudorandom Number Generator

Liu Feng

ShiJiaZhuang University of Economics, He Bei ShiJiaZhuang 050000, China

Gao Xiaoxing

Shijiazhuang University of economics, Hebei, Shijiazhuang 050031, China

**Abstract**—Random number sequences and RNGs play an important role in trusted computing environments and cryptographic applications. For example, we use random numbers in the generation of keys in TPM. In some web protocols, random numbers are applied to resist replay attacks. It is necessary to guarantee the quality of RNGs and their random sequences because deterministic factors are likely to be involved in the generation process. If a random number generator is not designed carefully, then the output number sequences may become predictable and bring high security risks. Thus, the design of random number generators that produce high-quality random number sequences has been a hot research topic in these decades. Recently, with the development of resource constrained environments, the demand of lightweight random number generators dramatically increases. People prefer to use the random number generators with extreme high efficiency in the on-the-fly applications. This will affect the security performance of the generators. In this paper, we design a random number generator which meets the current lightweight requirements in the resource-limited environments. Our design is originally based on a lightweight block cipher, and applies the property of random looking output of block cipher to the random number generators. We combine a traditional encryption mode with a novel structure for the random number generator, so that the trade-off between security and efficiency can be made perfectly. We also take a comprehensive security evaluation for our random number generator.

**Index Terms**—Random Number Generator; Random Numbers; Lightweight; MIBS; CBC Mode

## I. INTRODUCTION

In various modern cryptographic applications, random sequences and random numbers have been used widely. Weak random sequences and random numbers can destroy perfect cryptographic protocols and cryptographic algorithms even they are well designed. In typical cryptographic applications, randomness is produced by generating a sequence of independent uniform variants (usually real-valued between 0 and 1, or integer-valued in some interval) and transformed in an appropriate way.

In general, high-quality random number sequences have various distributions, long cycles, and sequence-independent characteristics. Quality inspection methods

have to follow the run-length, uniformity, and the independence of the sequences. These methods include a series of tests, correlation spectral analysis, and ENT (a performance testing program for random numbers) and so on.

Many applications need high-quality random sequences and random numbers. For instance, in hash functions, the initialization vectors are regarded as random. In secure protocols, the fresh nonce is thought to be random according to the different security requirements. In digital signature system, the variables and parameters are assumed to be random. When generating the session keys and the salts, which are hashed based on some rules, the randomness is also required.

The methods for generating random numbers have a range of types. From the mechanism, they can be divided into two kinds: mathematical and physical methods. The random number sequences generated by them are called pseudo-random numbers and true random numbers, respectively. The former one is easily cracked, while the latter one is difficult to crack, because they are taken from the real random sources of the physical world. However, this does not mean that the quality of these random numbers generated based on the true random sources is high, depending on how to use these algorithms to generate true random sources.

On the contrary, many mathematical methods are used to produce random numbers with better quality. Therefore, the combined mathematical and physical methods may produce high-quality true random numbers. From the point of implementation, there are different software-based and hardware-based methods.

Compared to the pseudo-random number generators, the study of true random number generator is still quite preliminary. Designing a true random number generator includes two steps. First, we need to get a true random source. Then, we obtain true random sources and true random numbers according to specific mathematical methods. True random sources exist widely in the real world, such as the arrival time of computer network IP packet, the random noise, the current second-level computer clock, the keyboard response time, the thermal noise, and the processing information of operating system.

There are also many other kinds of methods for generating random number sequences, such as entangled photon pairs, the photon polarization. This method can obtain by calling the system function or a hardware circuit to achieve the sources. For example, in [1, 2] the authors use the hardware circuitry to generate thermal noise. This is by building a chaotic circuit to generate a random sequence. In [3, 4] a method doing a system call is adopted to obtain the random nature of system processes and threads. To sum up, the methods to generate true random sequences with a true random number source are various. Among these, the simplest method is to directly use the sequences to generate the parity characteristics of 0-1 true random sources. In order to increase the randomness of a sequence, random numbers are often produced on the 0-1 sequences through a series of transformations, such as normalization, nonlinear mapping, displacement and encryption. For example, in [5] the authors use some really random sources that can be obtained from the computer. The random bits are stored in a buffer, and continue to perform arithmetic operations while using CRC-32 polynomial for a better blending buffer. When people request a random number sequence, the random numbers are read from the buffer sources. Firstly, we calculate the content of the buffer by MD5 or SHA hash functions [6]. Then, the MD5 or SHA values are written into a buffer to make sure that the feedback to the next buffer is not the same value as the previous one. The output values that are written into a binary user have a feedback for hiding the contents of the buffer. A random number sequence generated by this method has good statistical analysis for randomness [7].

Generation of true random sources require to consider the feasibility, economy, speed, and other issues. Some of the hardware circuit has an easier description, but it is difficult to achieve [8]. Some of the methods require higher cost. Some of the random sources must be related to human-computer interaction such as the randomness keyboard and mouse mentioned in [9]. The randomness is interrupted when there is no interaction. In this case, the randomness of random sources is greatly reduced. Some randomness measured by the incident frequency rate is very low [10], so the reliability of such random sources for generating random number sequences may be relatively low, which is not suitable for demanding real-time applications.

The above discussed the true random number generation method. In addition, a method that cannot be overlooked is based on nonlinear principle to produce high-quality random numbers, such as those based on chaotic map [11], cellular automata [12], and other principles of fractal random number generator [13]. Confidentiality chaotic system is better, because it is very sensitive to initial conditions, even in the case when the mapping is intercepted. As long as decimal digits of the seeds go enough large, the possibility to break the sequence is still relatively low. However, because such systems are inherently uncertain, if the recursive formula and one state are cracked, the subsequent state will be

cracked. Therefore, the requirements for the principle based on chaotic random number generator: (1) seeds accuracy is high enough; (2) mapping function is complex enough. You can change the parameters by random chaotic mapping to improve the chaotic complexity. In general, the chaotic mapping is still certain, but the literature is given a kind of uncertain chaotic mapping. It does not exist iterative relationship or depend on the initial conditions, which laid the foundation for producing high quality random numbers based on the principle of non-deterministic chaos.

Cellular automata (CA) are also used to generate random numbers [14]. CA has massively parallelism, local interaction, simplicity of cell structure and other excellent feature, which makes it very suitable for fast and efficient hardware implementation. Wolfram is the first one to apply CA to the random number generator [15]. His in-depth study of the dynamics of a uniform rule 30 and the application of the random number generator, and later CA-based non-uniform random numbers generation becomes the mainstream. Work in this area could refer to the literature.

In addition to the above methods, a combination of the random number generator is also a hot research topic. Press et al point out that double randomization technique is an important method to obtain high-quality random numbers. Another analysis about one-dimensional migration theory also verifies this idea.

#### A. Our Contribution

In this paper, according to the current lightweight requirements in the network security protocols and cryptographic applications, we design a random number generator which meets the security demands while still possesses high efficiency performance. We firstly propose a design of random number generator, whose creative advantage is to use a lightweight block cipher as the basic modules in our scheme. Using a lightweight block cipher as the primitive can make our random number generator meet the current lightweight requirements in the resource-limited environments. As we know, the output of a block cipher usually has a random looking so that the cipher messages cannot be predicted. A block cipher is to output the number sequences in a uniform and random way.

We also combine a traditional encryption mode with a novel structure for the random number generator. The traditional mode is widely-used and has been proved secure both theoretically and practically. Based on our novel structure, the advantage of the encryption mode can be maximized. Our design makes a perfect trade-off between security performance and efficiency performance, so that the output sequences of our generator can be regarded as highly random.

We carry experiments and theoretical analysis to evaluate the security of our random number generator as comprehensive as possible. The empirical and theoretical results both demonstrate the security of our design.

### B. The Organization

We organize our paper in the following way. In section 2, we give some preliminary for our work. We give the currently well-known random number generators in 2.A. Then, in section 2.B we give the mathematical background about the distance between two distributions. This is a basis for further measuring the quality of our random number generator. In section 3, we introduce our random number generator. Section 3.A is a general introduction of our generator. Section 3.B describes the lightweight block cipher MIBS, which is the core module in our random number generator. We discuss different layers in MIBS, and the key schedule of MIBS. In this section, we also introduce the working mode we use for performing MIBS. In section 3.C, we propose a new structure of the scheme of our random number generator. In section 4, we measure the security level of our random number generator, and compare the results with the random number generators mentioned in Section 2. In section 5, we conclude this paper.

## II. PRILIMINARY

### A. Existing Random Number Genertors

In practical simulations, we usually need random numbers or sequences possessing various distributions and different ranges. For the procedure of stochastic simulating generation, a reliable source for randomness is needed. For example, the move trace of a mouse, precise timing of Geiger counter clicks, the click time of a mouse, the thermal noise in some specific electronica circuit, and so on. In this section, we introduce two well-used random number generators that are related to this article.

#### 1) Three Shift Registers (SHR3)

We define the SHR3 as follows [16]:

$$a_1 = (y_i \ll m) \oplus y_i$$

$$a_2 = (a_1 \gg n) \oplus a_1$$

$$y_{i+1} = (a_2 \ll c) \oplus a_2$$

$\oplus$ :Xor

$a_1, a_2, y_i, y_{i+1}$ : n-bit words

$\ll$ : left shift

$\gg$ : right shift

The period of a SHR3 with nonzero seed is proved to be the maximum if and only if the characteristic polynomial is primitive.

In practical situation, if a triple set (m,n,c) is suitable for a SHR3 is examined by computing the primitive of the characteristic polynomial of the matrix. Previously a set of SHR3s with increasing periods is discovered. Two matrices are required for above operations.

$$L = \begin{pmatrix} 0 & 0 & 0 & & 0 & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & & 0 & 0 & 0 \\ \vdots & & \ddots & & \vdots & & \\ 0 & 0 & 1 & & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} 0 & 1 & 0 & & 0 & 0 & 1 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & & 0 & 0 & 0 \\ \vdots & & \ddots & & \vdots & & \\ 0 & 0 & 0 & & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & & 0 & 0 & 0 \end{pmatrix}$$

Note that

$$x_{i+1} = x_i T, \text{ where } T = (I + L^a)(I + R^b)(I + L^c)$$

Here,  $x_i$  is regarded as a column vector of m entries, and I is an identity matrix. As we showed above, L is an matrix which is filled with 1 in some predetermined positions and filled with 0 in the other positions. R is the matrix which is filled with 1 in all upper sub-diagonal entries and 0 in the other positions. R is regarded as the operation of the bitwise right shift, where L is for the operation of the bitwise left shift.

#### 2) Additive Lagged Fibonacci Generators (ALFGs)

In this sub-section, we consider the additive lagged Fibonacci generator (ALFG) [17], as follows:

$$B_n = B_{n-p+q} + B_{n-p} \bmod 2^q$$

where the maximal value of  $2^{q-1}(2^p - 1)$  can be acquired for q, q is the size of the word, under the following condition:

$$y^c + y^p + 1 \bmod 2$$

The above equation is a primitive polynomial in mod 2. That is,  $y^c + y^p + 1$  should be primitive. Note that a reciprocal of a primitive polynomial is also primitive. Given the polynomial  $A(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$ , its reciprocal polynomial is  $x^n A(x^{-1})$ . These two polynomials are reciprocal of each other and primitive.

### B. Mathematical Background

In order to demonstrate that our random scheme can act as a random number source, we need to show that it is difficult to distinguish our design with the real random number sequences. That is, given the uniformly generated sequences, the distributions of some specific statistics computed from our generalized number sequences cannot be distinguished from the real random ones. Thus, here we introduce the statistical and mathematical preliminary for measuring the differences between two probability distributions.

We denote by the field with two elements and  $F_2^m$  denotes the m-dimensional vector space over  $F_2$ . Let X be a discrete random variable in and

$p = (p^0; p^1; \dots; p^{2^m-1})$  be the probability distribution of X such that

$$p^l = \Pr(X = l),$$

where  $l \in F_2^m$ .

The function  $f = (f_1, f_1, \dots, f_1): F_2^n \rightarrow F_2^m$  is called a vector Boolean function, where

$f_i : F_2^n \rightarrow F_2$  is a Boolean function.

**Definition 1 [18].** Let  $p = (p^0; p^1; \dots; p^l)$  and  $q = (q^0; q^1; \dots; q^l)$  be two discrete probability distributions with sample space  $S$ . Then, the capacity between and is

$$C(p, q) = \sum_{l \in S} \frac{(p^l - q^l)^2}{q^l}$$

In the case that  $q$  is the uniform distribution, the capacity is denoted by  $C(p)$ . Let  $Y$  be a Bernoulli( $p$ )-distributed random variable which takes values in  $\{0,1\}$  such that  $\Pr(Y = 0) = p$ . Then the correlation of  $Y$  with zero is defined as

$$C(Y) = 2\Pr(Y = 0) - 1 = 2p - 1$$

And, the bias denoted by is equal to  $c(Y)/2$ . Let  $Z$  be an  $m$ -bit random variable with probability distribution  $p$  and  $a \in F_2^m$ . Then, we have

$$C(a \cdot X) = \sum_{l \in F_2^m} (-1)^{a \cdot l} p^l$$

For the  $m$ -bit random number sequences generated by a pseudo-random or real random number generator, we have the following proposition about their distributions. This proposition proves that the probability distribution  $p$  of  $m$ -bit random variable  $X$ , taking values from  $F_2^m$ , is computed by the correlations of  $a \cdot X$ , where .

**Proposition 1.** Let  $X$  be an  $m$ -bit random taking values from  $F_2^m$  variable with probability distribution  $p$ , then

$$p_l = 2^{-m} \sum_{a \in F_2^m} (-1)^{a \cdot l} c(a \cdot X), \forall l \in F_2^m$$

**Proposition 2.** Let  $X$  be an  $m$ -bit random variable taking values from  $F_2^m$  and  $p$  be its probability distribution. Then, the capacity of  $p$  is

$$C(p) = \sum_{a \in F_2^m - \{0\}} c(a)^2$$

To estimate the distinguishing ability of two distributions, we can use the Chernoff information  $D$  [18].

**Theorem 1 [18].** Let  $\text{BestAdv}$  be the best advantage for distinguishing probability distribution  $p$  from probability distribution  $q$ , using  $N$  samples. We have the following relations:

$$1 - \text{BestAdv} = 2^{-ND(p,q)}$$

### III. OUR NEW RANDOM NUMBER GENERATOR

#### A. Introduction of Our New Random Number Generator

In this section, we introduce the general structure of our random number generator. This random number generator scheme is based on the primitive of symmetric encryption, including stream ciphers and block ciphers. We use the efficient and succinct modules in these ciphers, and iterate these modules by enough rounds so that the input random seed can be generated as random output sequences which act in the similar way with the true random sequences. In cryptographic field, it is well-known that an important property for both the stream ciphers and block ciphers is that their outputs behave like the uniform and random number sequences. That is, we

cannot tell the differences between the output of the ciphers and the output of a truly random permutation. This fact is the primary basis of our design idea. In the following, we start with introducing the block cipher modules used in our random number generator design, and then we give the overall structure of our design in a high level.

#### B. Modules of Our New Random Number Generator

We use the primitives of a lightweight block cipher, namely, MIBS. MIBS is a block cipher using conventional Feistel structure, focusing on the applications of resource constrained environments. MIBS has a 64-bit block size supporting 64-bit and 80-bit keys and iterates 32 rounds for both key sizes. The round function  $F$  of MIBS is an SPN composed of an XOR layer with a round key, a layer of  $4 \times 4$ -bit S-Boxes (S layer), and a linear transformation layer (P layer), in this sequence. The components of the encryption process involved in  $F$  are explained as follows. Note that all internal operations in MIBS are nibble-wise, that is, on 4-bit words.

**Key addition:** In each round  $i$ ,  $1 \leq i \leq 32$ , the 32-bit input state  $s_i$  to the  $F$  function is XORed with the round key  $K_i$ , that is  $\{s_i\} \oplus \{K_i\}$ , where  $\oplus$  denotes XOR.

**Non-linear Layer:** After key addition, the state is split into eight nibbles and identical  $4 \times 4$ -bit S-Boxes (see Table I are applied in parallel.

TABLE I. THE S-BOXES OF MIBS

x	S(x)
0	4
1	f
2	3
3	8
4	d
5	a
6	c
7	0
8	b
9	5
a	7
b	E
c	2
d	6
e	1
f	9

**Linear transformation layer P:** The input  $(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$  is transformed into its output  $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8)$  by the following relations:

$$z_1 = y_1 \oplus y_2 \oplus y_4 \oplus y_5 \oplus y_7 \oplus y_8$$

$$z_2 = y_2 \oplus y_3 \oplus y_4 \oplus y_5 \oplus y_6 \oplus y_7$$

$$z_3 = y_1 \oplus y_2 \oplus y_3 \oplus y_5 \oplus y_6 \oplus y_8$$

$$z_4 = y_2 \oplus y_3 \oplus y_4 \oplus y_7 \oplus y_8$$

$$z_5 = y_1 \oplus y_3 \oplus y_4 \oplus y_5 \oplus y_8$$

$$z_6 = y_1 \oplus y_2 \oplus y_4 \oplus y_5 \oplus y_6$$

$$z_7 = y_1 \oplus y_2 \oplus y_3 \oplus y_6 \oplus y_7$$

$$z_8 = y_1 \oplus y_3 \oplus y_4 \oplus y_6 \oplus y_7 \oplus y_8$$

**Key Schedule:** MIBS has two versions of key size, 64 bits and 80 bits. In our application, we take the original input seed of the random number generator as the input key of MIBS.

**Cipher mode:** As we know, the input size and the output size of MIBS are fixed, while the random number generator is supposed to output random sequences with arbitrary size. Thus, we need to use MIBS under some suitable mode so that we can extend the fixed-length input into the flexible-length output sequences. Fig. 1. is the mode we choose.

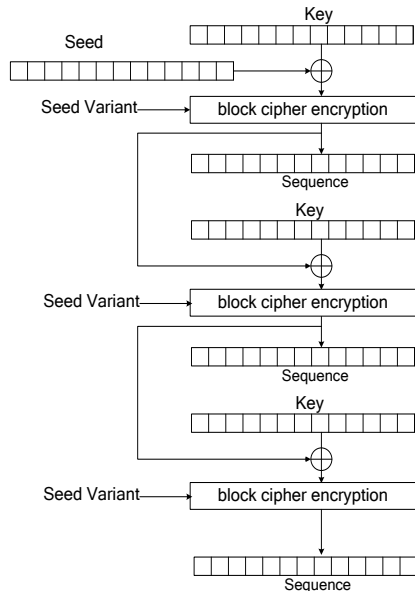


Figure 1. The structure of CBC mode

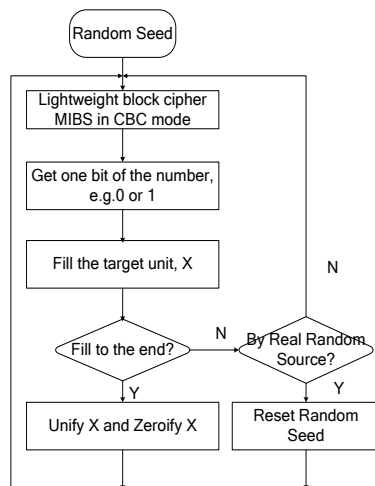


Figure 2. Our structure of random number generator

The cipher-block chaining (CBC) mode of operation was developed by IBM in 1976. For MIBS in CBC mode, each block of input message is XORed with the previous output message before going through MIBS. In this way, each output message depends on all input message encrypted up to that point. To make each message unique, an initialization vector must be used in the first input message. In our application, we change the mode like this.

We use the truly random seed as the initialization vector. The input message blocks are replaced by seed, seed+1,....., seed + i, subsequently. The length i is dependent on the length of the output random number sequences. The key is a variant generated from the random seed. The output messages are now the required output random numbers.

CBC is the most commonly used mode of operation. Its main flaws when applied for block ciphers are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size. However, in our application, these flaws change into the advantage since we definitely require the sequential sequences.

### C. General Structure of Our New Random Number Generator

Figure 2 shows the structure of our random number generator scheme. The flow steps are as follows.

- Input the random seed with a user client key. The key can be chosen randomly but doesn't need to be kept secret.
- The rand seed with the key goes through the lightweight block cipher MIBS, as we mentioned in the above section. MIBS is operated in the CBC mode, as in Fig. 1.
- Take the most significant bit from the output of the CBC lightweight MIBS, e.g., 0 or 1.
- Fill in a target unit, X. The length of X should be enough to provide unpredictability.
- If X is filled, check whether it is from the truly random source.
- If the X can be regarded as from the truly random source, then reset the random seed for the next generation.
- If the X cannot be regarded as from the truly random source, then regenerate the number from Step b).
- Unify X and put the value of X as zero.

## IV. SECURITY EVALUATION OF OUR RANDOM NUMBER GENERATOR

### A. Experimental Evaluation Based on the Theoretical Results

Firstly, we measure our random number generator scheme based on the theoretical results of a chaotic map, which is in the following form:

$$Y = \sum_{i=r}^n \alpha \pi \sin \frac{n!}{r!(n-r)!} \quad (1)$$

It is the exact solution of

$$z = \sum_{i=r+1}^{n-1} \kappa \pi \cos \frac{(n+1)!}{r!(n+1-r)!} \quad (2)$$

When z takes integers, fractions, rational numbers, irrational numbers or other types, the corresponding return map is totally different. When z is an irrational number, its first return map is not regular, and the formula (1) is different from the general uncertainty of

chaotic map. It does not depend on the initial conditions, but only on the parameter  $\pi$ ,  $z$ , and the value of the variable  $n$ . Note that  $n$  is dependent on a degree constant  $\theta$ . Because of the unpredictable characteristics, it can be used to generate random numbers.

In formula (1)  $\pi$  can be changed dynamically. For example, according to a simple example chaotic mapping to determine, each time a certain number (set  $M$ ) of random numbers is generated, the value of  $\theta$  could change accordingly. Meanwhile reset the  $n$  in formula (1) to 0. However, this type of random number sequence obtained from formula (1) has slightly inferior uniformity, and it could be improved by conversion to formula (2).

This approach will be applied to the program in section 3. When the real random source is generated, resetting the seed becomes changing the value of  $\theta$ , and resetting the value of  $n$ . In addition, we need to change  $\theta$  and  $n$  every  $M$  times. Since  $n$  is often reset,  $n$  will not be too large. Therefore, the calculation in formula (1) is not going to be very complicated. Moreover, studies have shown that the randomness of the sequence generated by changing the value of  $\theta$  is exactly the same as that generated by not changing  $\theta$ .

Based on our scheme, sampling 262144 random numbers (Note that here  $N$  is 32,  $M$  is 500) cost 40s in a PC, in which 117 times of resetting is performed. Use this as a sample for testing the uniform property and independence of the overall delegates. Note that we test the uniform property in the  $[0,1]$  uniform distribution.

Use  $\chi^2$  fit test method to test the uniform property of the samples' distribution. The samples are divided into 24 intervals of equal width. We calculate the number of samples  $S_n$  within each interval  $i$  (see Table II).

TABLE II. THE NUMBER OF RANDOM SEQUENCES IN DIFFERENT INTERVAL

Interval $i$	$S_n$	Interval $i$	$S_n$
1	20641	7	14234
2	19862	8	20451
3	20041	9	21456
4	14783	10	28571
5	15972	11	28746
6	12482	12	18963
Interval $i$	$S_n$	Interval $i$	$S_n$
13	13578	19	20787
14	16872	20	21470
15	19872	21	20334
16	18753	22	20174
17	18963	23	20729
18	19857	24	19871

For a uniform distribution, the probability of each random falling in the  $i$ th interval is  $\Pr(S_n) = 1/24$ .

According to  $\chi^2$  fit test method, use statistic

$$Q = \sum_{j=1}^l \frac{(n_j - np_j)^2}{np_j}$$

In the above statistic,  $l$  is the number of intervals,  $n$  is the number of samples, we can compute that  $Q$  equals 28.02. Taking  $\alpha = 0.05$ , looking up the  $\chi^2$  distribution table with  $k = l - 1 = 23$ ,  $\chi^2 = 0.00523^2 < 2$ . Therefore,

the assumption of significant level  $\alpha = 0.05$  should be accepted.

It is reasonable to believe the distribution of the integral represented by the samples obeys the  $[0, 1]$  uniform distribution.

In addition, we perform the test using the KS method proposed by Kolmogorov (a homogeneous Detection method),  $a = 0.0729 < D1$  and  $b = 0.1267$ . Therefore, the assumption of uniform distribution is acceptable. By theoretical and experimental analysis, it is shown that the random sequences generated by our proposed random number generator meet the characteristics of a uniform distribution on the interval of  $[0, 1]$ .

### B. Comparison of Our Random Number Generator with SHR3, ALFGs and Truly Random Generator

In this section, we compare the output sequences of our generator with the three random number generators we introduced in Section 2. We generate 20000 30-bit random numbers for each generator, and then use several statistical randomness tests on the output sequences. Table III is the number of random sequences passing the tests.

TABLE III. COMPARISON OF PASSING RANDOM NUMBERS

	ours	SHR3	ALFGs	Truly
SAC test	19831	10162	18557	11148
Frequency test	15434	14512	14585	14758
Run test	16649	14527	15637	14274
Autocorrel-ation test	19992	17788	15863	12457
Binary matrix test	16381	17287	16987	17832
Random walk test	15523	18588	15893	17458
Birthday test	18828	19358	14789	12563

TABLE IV. THE COMPARISON OF EXECUTION TIME

ours	SHR3	ALFGs	Truly
0.05s	0.07s	0.07s	0.04s
0.04s	0.05s	0.08s	0.08s
0.06s	0.09s	0.05s	0.07s
0.05s	0.04s	0.08s	0.05s
0.04s	0.06s	0.04s	0.08s
0.06s	0.05s	0.05s	0.05s
0.06s	0.08s	0.06s	0.07s

From Table II, we can find that our random number generator has a relatively higher passing ratio for the output sequences. Note that, all results are measured under the same  $p$ -value, meaning the same range. The testing results of our scheme are especially good for the SAC test and the Autocorrelation test. And for the Frequency test and the Random walk test, the passing sequences from our output are lower than other tests. The SHR3 performs best for the birthday test and performs the worst in the Frequency test and Run test.

### C. Comparison of Efficiency Performances of Our Generator, SHR3, ALFGs and the Truly Random Generator

In this section, we compare the execution speed of our generator with other three ones mentioned in Section 2. The results are as follows.

From Table IV, we can conclude that our random number scheme has a stably high efficiency. We repeat

the experiments seven times. Each time we generate 10000 50-bit numbers randomly, and record the execution time.

## V. CONCLUSION

Random number sequences and RNGs are the fundamental primitives in network security environments and cryptographic protocols. I.e., they can be used in the generation of keys in trusted platform. Also, in some key exchange protocols, random numbers are used to provide refresh properties. In practical, it is important to make sure the random number sequences generated by some random number generator are random enough. This means that there are no deterministic factors that can be detected in the generation process.

In this paper, we deal with a hot research topic, on the design of random number generators that produce high-quality random number sequences.

Recently, with the development of resource constrained environments, the demand of lightweight random number generators dramatically increases. People prefer to use the random number generators with extreme high efficiency in the on-the-fly applications. This will affect the security performance of the generators.

In this paper, according to the current demands of lightweight application in network security, we design a random number generator which satisfies the security requirements while still executes with a high speed. We exploit the advantage of a lightweight block cipher called MIBS in our design. This facilitates our random number generator to follow the lightweight demands in the resource-limited environments. As we know, the output of a block cipher should be random enough so that the pattern of cipher messages cannot be discovered.

We also design a novel scheme for our random number generator. The traditional CBC cipher mode is applied and has shown a high efficiency practically. Based on our novel structure, we can have a perfect trade-off between the efficiency performance and the security performance.

Finally, we make theoretical analysis to estimate the security of our random number generator and compare the results with the SHR3, ALFGs and a truly random number generator. The empirical and theoretical results show that the output sequences of our generator can be regarded as highly random and the execution speed of our scheme is quite fast.

## REFERENCES

- [1] Xin Qian, Zeng Yang, Zhang rights, such as true random number generator system modeling and simulation, *Journal of System Simulation*, 2005, 17 (1) pp. 53-56.
- [2] Wang Lai, Song-Qiang Liu design and implementation of a true random number generator *Nuclear Electronics & Detection Technology*, 1998, 18 (6) pp. 452-455
- [3] Knuth D E. The Art of Computer Programming. Addison-Wesley, 1981.
- [4] Press W H, Flannery B P, Teukolsky S A, et al. Numerical Recipes in Fortran: The Art of Scientific Computing. Cambridge University Press, 1996.
- [5] Hortensius P D, Mcleod R D, Card H C. Parallel Random Number Generator for VLSI Systems Using Cellular Automata. *IEEE Transactions on Computers*, 1989, 38(10) pp. 1466-1473.
- [6] M. Gude, "Concept for a High-Performance Random Number Generator Based on Physical Random Noise," *Frequenz*, v. 39, 1985, pp. 187-190.
- [7] P. C. van Oorschot and M. J. Wiener, "Improving implementable meet-in-the-middle attacks by orders of magnitude," *CRYPTO '96*, Springer-Verlag, 1996.
- [8] R. P. Brent. Note on marsaglia's xorshift random number generators. *Journal of Statistical Software*, 11(5) pp. 1-4, 2004.
- [9] George Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14) pp. 1-6, 2003.
- [10] George Marsaglia and Wai Wan Tsang. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3) pp. 1-8, 2002.
- [11] George Marsaglia and John Marsaglia. Evaluating the Anderson-Darling distribution. *Journal of Statistical Software*, 9(2) pp. 1-5, February 2004.
- [12] George Marsaglia, Wai Wan Tsang, and Jingbo Wang. Evaluating Kolmogorov's distribution. *Journal of Statistical Software*, 8(18) pp. 1-4, 2003.
- [13] W. W. Tsang, L. C. K. Hui, K. P. Chow, C. F. Chong, and C. W. Tso. Tuning the collision test for power. In *Proceedings of the 27th conference on Australasian computer science*, pp. 23-30. Australian Computer Society, Inc., 2004.
- [14] Elaine Barker and John Kelsey, NIST Special Publication 800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*.
- [15] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Sponge-Based Pseudo-Random Number Generators, *CHES 2010*, pp. 33-47, 2010.
- [16] Yongge Wang. Resource bounded randomness and computational complexity. *Theoret. Comput. Sci.*, 237 pp. 33-55, 2000
- [17] Yong Song, Xianfu Chen, Haidong Yao. Discussion on High-quality RNG and Scheme of True RNG. *Computer Engineering*. Volume33, Number 2, pp. 71-73. 2007
- [18] Thomas Baignères, Pouyan Sepehrdad, Serge Vaudenay. Distinguishing Distributions Using Chernoff Information. *LNCS Volume 6402*, pp. 144-165. Provable Security, 2010.