Proposed Pseudo Random Generator Based on RC5 Block Cipher

Ashwaq Talib Hashim[1], Zaid Mundher Radeef [2]
[1]Control and Systems Eng. Dept., University of Technology, Baghdad, Iraq.
[2]Computer Science Dept., University of Technology, Baghdad, Iraq.
e_mail:60102@uotehnology.edu.iq, zaidmalani@gmail.com

Abstract *— Digital cryptography relies greatly on randomness in providing the security requirements imposed by various information systems. Just as different requirements call for specific cryptographic techniques, randomness takes upon a variety of roles in order to ensure the proper strength of these cryptographic primitives. This paper presents a 64-bit random number generator with a high generation speed and a good random sequences by employing the RC5 encryption algorithm round functions with some changes in the usage of the key, where two keys have been used with an initial vector to start the sequence generator. This generator can be used as a keys generator for any security system or to generate an intermediate password for signing process, and it can be adapted in many other applications, such as random pattern generation, games, etc. Examining the sequences shows great results in terms of randomness and security by achieving a difference near 0.5, this value is the best result, and a high undependability on the previous generated sequences by also achieving a difference more than 0.5, frequency, serial and run test came out with a near optimality results.*

Index Terms *— Pseudo random generator, Randomness, Block cipher, Cryptography, Security.*

I. INTRODUCTION

Block ciphers are the most popular cryptographic primitives; due to the standardization of DES followed by AES, but also due to the fact that block ciphers constitute some of the fundamental building blocks for pseudorandom number generators, stream ciphers, hash functions and message authentication codes [1].

Most block ciphers apply a simple encryption function, round function, iteratively a certain number of times - called rounds, using round keys derived from the initial secret key using a key expansion algorithm. There are two main types of iterated ciphers: Feistel ciphers (e.g. DES, Blowfish, and CAST-128) and Substitution-Permutation network ciphers (e.g. AES and Serpent) [2].

Security in all of its aspects requires a high degree of randomness to achieve the desired goal, a strong hard to crack security system, while the build of such a random generator needs to deal no impact on the performance, while still achieving the best results [3]. An automated security solution also needs a good random generator, without the need for human intervention to supply seeds to the generator [4].

Random generator is a computational algorithm to generate a sequence of bits, numbers or characters with unpredictable chance to guess it only randomly [5].

Many applications lead to the need of generating random data, such as cryptography, secret key generator or authenticating system to make the system hard to attack or predict [6].

The environment where the random generator will be used also adds some limitations to the random sequence, in cryptography as an example, it is desirable that the generated number should neither be repeated nor discovered, or the system will be vulnerable. In simulation as an example the sequence is desired to be repeated not only an individual number. The true random number generator uses a source

Received 13 April 2017; Accepted 3 October 2017
 ISSN 1811-9212

with high entropy (the amount of uncertainty about an outcome), for example; the coin flip, rolling dice and drawing a card from a deck. These examples have a high degree of entropy, because it is impossible to predict the result. True random disadvantages are that they are low rate production systems, and they are implemented using some sort of hardware devices to roll the dice or to flip a coin and another to record the results [7].

The true random number generator needs real world phenomena, on the other hand, the pseudo random number generator does not need any real world phenomena to generate a random sequence. This kind of generators uses seeds such as initial value to start producing pseudo random numbers [8]. Some of the systems use time, date, and location on the pointer as an initial value, some needs user interaction with the process to get the initial value, this initial value has a great impact on entropy. Some of the well-known pseudo random generators are lagged Fibonacci, feedback shift registers and many other generators.

The rest of the paper is organized into 5 sections, some of the related works are illustrated in Section II, an overview of RC5 algorithm has been described in Section III, Section IV gives a full description of the proposed system along with the achieved results in Section V, and finally the conclusion is pointed out in Section VI.

II. RELATED WORKS

1. In 2004, Moni Naor and Omer Reingold introduced the Naor–Reingold pseudorandom function and achieved a good statistical distribution in almost all vectors [9].
2. In 2006, Ghusn Alban Ali Ahmed presented a random number generator based on the division algorithm [10].
3. In 2008, Christophe Petit, et al., introduced a Block Cipher Based Pseudo Random Number Generator Secure against Side-Channel Key Recovery, they add some noise and masks to the process of generation to make it impossible to revert the process and build the generator [11].
4. In 2014 Ashwaq T. Hashim and Loay E. George proposed a pseudo random generator dependent on a secret key and a timestamp. The secret information is distributed randomly into n segments [12].
5. In 2015, Alaa kadhim et al., presented the design and implementation of a system of keys generator with nonlinear random of output keys and large moment bits [13].

III. THE RC5 ALGORITHM

RC5 (Rivest Cipher version 5) is a symmetric key block cipher; its design makes it suitable for implementation in hardware and software. RC5 is a variable block size algorithm which works on 32, 64 or 128-bit block of data, variable number of rounds (0 to 255) and a variable key size (0 to 2040-bits). These options give a high flexibility in terms of performance and security.

The optimal configuration for both security and performance is 64-bit block, 128-bit key and 12 rounds simply written as RC5-64/12/16, using RC5-w/r/b as the configuration symbol where w is the block size, r is the number of rounds, and b is the key length in bytes.

The overall design of this algorithm is divided into 3 components: key expansion, encryption phase and decryption phase.

The main operations in these three components are:

1. Addition of words modulo 2^w
2. Bit-wise exclusive-OR of words
3. <<< Rotation symbol: the rotation of x to the left by y bits is denoted by $x <<< y$.

The simple design of RC5 makes it easy to implement, while the high degree of data dependency makes it hard for differential and linear cryptanalysis.

Received 13 April 2017; Accepted 3 October 2017

 ISSN 1811-9212

To implement the RC5 algorithm, first the input block is divided into two W-bit word blocks, let A and B represent these two words. The resultant two W-bit words are also placed in A and B.

Using the sub-keys array S[0]…..S[n] generated using key expansion algorithm(described later), the operations of the encryption rounds are:

A = A + S[0]
B = B + S[1]
for i = 1 to r do
A = ((A B) <<< B) + S[2i]
B = ((B A) <<< A) + S[2i + 1]
End loop

The output is in A and B W-bit words.

The decryption phase of the RC5 algorithm is just the opposite operations of the encryption, as described below:

For i = r down to 1 do
B = ((B − S[2i + 1]) >>> A) A
A = ((A − S[2i]) >>> B) B
B = B − S[1]
A = A − S[0]
End loop

The RC5 encryption/decryption algorithms are illustrated as shown in Fig.1 and Fig.2, respectively.

The key expansion algorithm expands the secret key K 'provided by the user to the required number of keys (2 × number of rounds) and the keys are collected in array S. The key expansion algorithm uses two constants and works in three simple operations.

The key expansion algorithm uses two word binary constants Pw and Qw. For w = 32, these constants are given below in binary and in hexadecimal.

Pw = P16 = 1011 0111 1110 0001 = B7E1
Qw = Q16 = 1001 1110 0011 0111 = 9E37

The user provided secret key is firstly converted from bytes to words by copying the secret key K[0...b-1] into an array L[0...c-1] of c = [b / u] words, where u=w/8 is the number of bytes/word, as follows:

c = [max(b, 1) / u]
For i = b - 1 downto 0 do
L[i / u] = (L[i / u] <<< 8) + K[i]

Then array S is initialized as follows:

S[0] = Pw;
For i = 1 to t - 1 do
S[i] = S[i - 1] + Qw

Then array S will be computed as follows:

i = j = 0;
a = b = 0;
do 3 * max(t, c) times:
a = S[i] = (S[i] + a + b) <<< 3
b = L[i] = (L[j] + a + b) <<< (a + b)
i = (i + 1) mod (t)
j = (j + 1) mod (c)

Received 13 April 2017; Accepted 3 October 2017
 ISSN 1811-9212

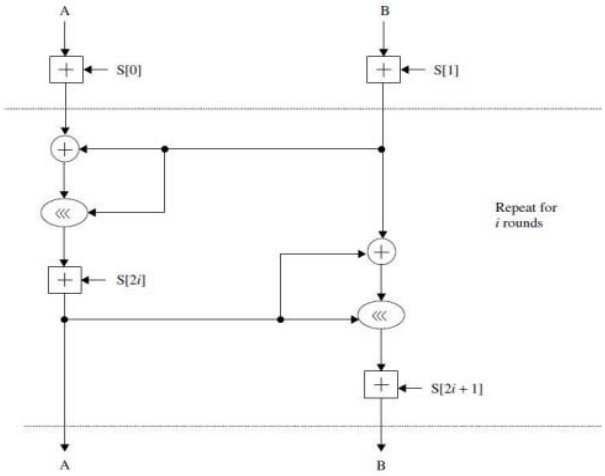

FIG. 1: THE RC5 ENCRYPTION ALGORITHM

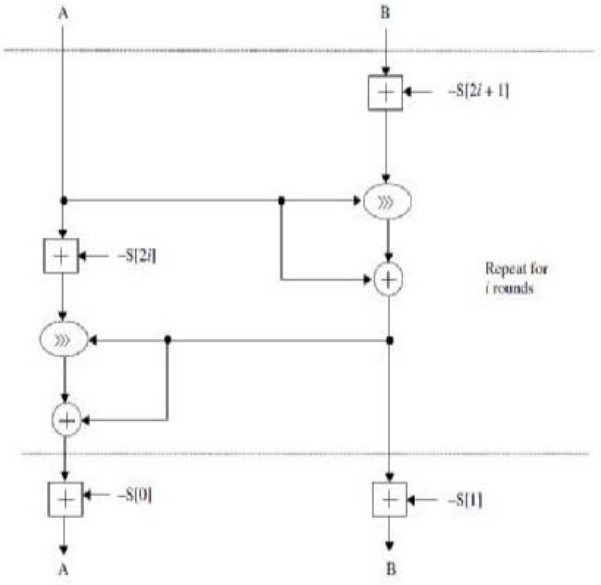

FIG. 2: THE RC5 DECRYPTION ALGORITHM

IV. PROPOSED SYSTEM

Cryptography relies mainly and mostly on the randomness, the rate of randomness affects the strength on any security system and adds a high rate of complexity to the system. RC5 block cipher based pseudo random number generator has been proposed. Fig.3 depicts the general structure of the pseudo random sequence generator where IV is the initial vector of the generator and the set (K, K') are the keys (seeds). The proposed pseudo random generator is based on using the rounds of the highly random block cipher algorithm RC5.

The KSG is a serial combination of two instances of RC5 block cipher placed into the cipher block chaining encryption mode. The input of the first RC5 is initialized to a public IV, and each block cipher is initialized with its own master key, denoted k_i and k'_i, respectively, these keys are playing the role of seed for the pseudorandom generator. *Fig.*4 shows the block diagram of KSG. This system generates 64 bit long random number as described in Algorithm (1).

As noticed in Fig.4, the x_i is the input to the first RC5, and the m_i is an intermediate. Then the output of the KSG is y_i.

Received 13 April 2017; Accepted 3 October 2017
 ISSN 1811-9212

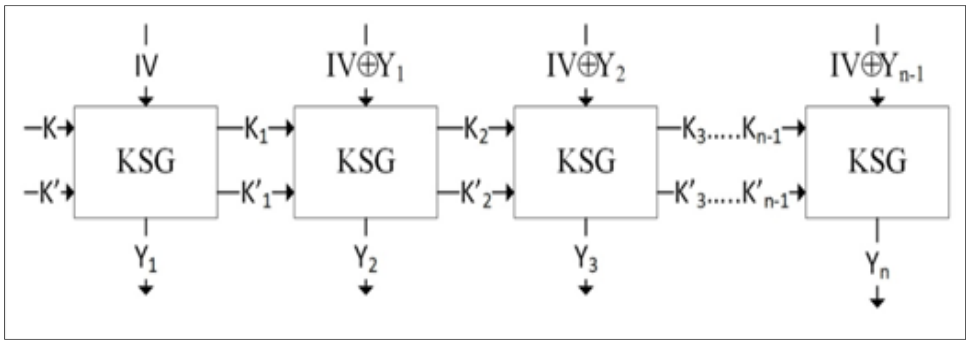

FIG. 3: THE PROPOSED PSEUDO RANDOM NUMBER GENERATOR

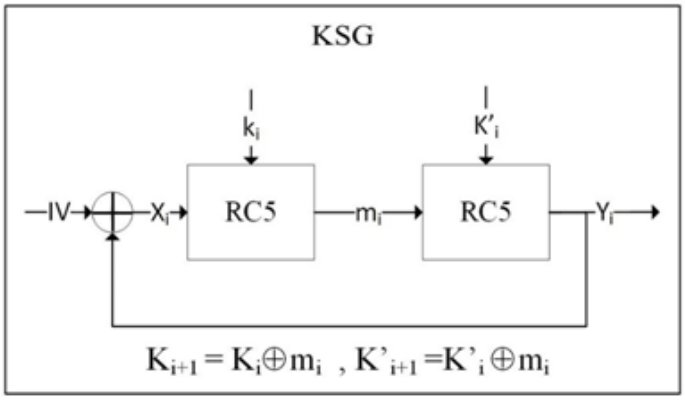

FIG. 4: THE PROPOSED KEY STREAM GENERATOR (KSG)

Algorithm (1) Proposed Pseudo-Random Generator	
Input: Key1 //64-bit master key number 1 Key2 //64-bit master key number 2 IV //64-bit initial random generator vector L //Sequence length	
Output: Sequence //Random Sequence	
Step1: Generate L random numbers based on IV, Key1 and Key2 For i =1 to L Input=IV m_i= RC5(Input, key1) yi= RC5 (m_i, key2) Input = Input ⊕ y_i Key1 = key1 ⊕ m_i Key2 = key2 ⊕ m_i Sequence [i] = y_i End loop	 (1) (2) (3)

V. EXPERIMENTAL RESULT

Pseudo-random number generator relies on two master keys and an initial vector to generate as many random numbers as needed. Many statistical tests were produced to test the randomness of a sequence, in general many sequences that are considered random maybe easy to predict.

Received 13 April 2017; Accepted 3 October 2017

ISSN 1811-9212

So it's important to test the generator to prove its efficiency, some of the well-known widely used NIST suggested tests are given below: [15]

a. Frequency test: It determines how many zeros and ones in number, the closer probability between them the better randomness achieved, the perfect ratio is 1:1 [15].
b. Runs test: This test checks the runs of identical consecutive bits in the sequence. This run should not be long [15].
c. Serial test: This test calculates the frequency of all possible patterns (00, 01, 10, 11) [15].
d. Difference test: This test runs on two generated sequences, and determines the bit's difference in the generated sequence, it is preferable that the difference should pass 0.5 of the sequence length [15].
e. Information entropy: Unpredictability degree of an information is measured by the information entropy. consider as an example a coin flip, the result of the flip is unknown in advance and unpredictable in terms of math, the entropy of such an example is very high, now let's flip the coin again, the result of the second flip is not related to the first flip and preserves its entropy. let's take another example a game of domino the first stone is unpredictable and achieves high entropy, but as the game goes on the stones become slightly predictable and entropy reduces, till it reaches a case where the next stone is 100% predictable and entropy vanishes [16].

In terms of security, it is preferable to get high entropy and preserve it as system goes on. Entropy can be measured by the following equation:

$$H = -\sum_{i=0}^{n} p(x_i) \log_2 p(x_i) \quad (1)$$

f. Avalanche affect: It is a desirable property in cryptography and hash function, in which changing few bits in text or key should do a huge change in the output (more than half of the result size is desirable to be affected) [16].

Table I shows first three samples generated using the specified initial vector and keys, while Tables II to VII show some of the tests result, such as the impact of each input on the generated sequence when changing one bit of each input, frequency of 0's and 1's, the difference between the generated sequences, the difference between the first sequence of each sample, longest run of 0's and 1's in each sequence, frequency test and serial test.

The speed of execution is 200000 sequence/sec or 12,800,000 bit/sec.

Table II shows the difference between the first generated sequences of the samples. we can find the avalanche effect of each generated sequence, as mentioned earlier, it is preferred to achieve a total difference of half or more the length of the sequence when one bit or more is changed in the input parameters, in the proposed generator the length is 64 bit so any difference near 32 is considered very good.

In binary random sequence generation algorithms it is preferable to acquire a close frequency of 0's and 1's to each other, while getting a frequency of 0's equals to frequency of 1's is considered the perfect situation, but the samples that have this property are a small portion of the entire population.

The results in Table III shows that the property of close frequency has been achieved even the perfect situation has been achieved in some samples and it shows that the difference between subsequent sequences is more than half of the bit length.

Table IV contains the results of the frequency test, one of the famous tests to check the randomness of a sequence is that; most of the results are below the threshold of randomness "3.84".

Another test of randomness is the serial test; Table V contains the result of the test, where all the samples passed the randomness condition, which is below the threshold of the test "5.99".

Table VII lists the information entropy of each generated sequence, for a binary sequence the best possible entropy to achieve is 1 which is the maximum value, while any value below 0.8 will be considered as worse, it's obvious from looking at Table VII that all the achieved values are very good.

Received 13 April 2017; Accepted 3 October 2017
 ISSN 1811-9212

Table VI contains the longest run of 0's and 1's in each generated sequence, and it shows good results, long run of a bit leads to a bad random sequence, still the acquired values are good.

TABLE I: THREE CONSECUTIVE RANDOM SEQUENCES GENERATED USING DIFFERENT INITIAL AND MASTER KEYS.

Sample no	Initial vector	Master Key 1	Master Key 2	First random sequence	Second random sequence	Third random sequence
1	0	0	0	100000010101 000010100011 111010010100 100101011100 111001001000 0010	001100010001 001001010011 001000110010 000000111011 011011111000 0101	111010000001 110011010101 110100100011 101011111100 001001011100 1110
2	0	1	0	010000110100 110000111001 111100100011 011100001111 011010110110 0100	011000111110 011101110110 011011010100 010010011110 011011110001 1011	110010111001 111111110111 110001000110 010010010000 001001000111 1011
3	0	0	1	100000011010 000100100000 111001100000 101001000000 100111001010 1100	000100100100 100100001001 111101011111 011101100001 110101110111 1101	100100101110 011110001101 001100100001 110100001111 100011001101 0000
4	1	0	0	101011001101 100000010110 110000101110 001111101100 111001111011 0111	100101101100 101100111011 111000011101 110101111101 011010111101 1100	000000110100 110101011101 011110110101 111101000011 100110000011 0011
5	100100111101 111000101000 101000000011 101100001001 110010101101 0010	011000110101 010110100101 000000010001 000010101001 110011101001 0001	001111110001 110111011110 010001110111 011110100001 111100011111 1111	011001011000 110100100001 001111111111 111101101001 000110110011 0010	001111010011 010010000111 110001110101 001100011001 001001111010 1110	010001101111 111110011000 110100010000 001010100100 001101010000 0110
6	100100111101 111000101000 101000000011 101100001001 110010101101 0010	001111110001 110111011110 010001110111 011110100001 111100011111 1111	011000110101 010110100101 000000010001 000010101001 110011101001 0001	000011011011 101011111100 101101011110 100100001010 001011010110 1001	001011110111 010000000001 010111000111 010001010111 111111000010 0011	000000011011 100000001001 100110101011 101100000110 100111001100 1100

TABLE II: DIFFERENCE BETWEEN FIRST GENERATED SEQUENCES IN THE SIX SAMPLES

Sample no	1	2	3	4	5	6
1	0	34	26	28	37	34
2	34	0	34	34	27	32
3	26	34	0	34	31	32
4	28	34	34	0	35	34
5	37	27	31	35	0	33
6	34	32	32	34	33	0

Received 13 April 2017; Accepted 3 October 2017

 ISSN 1811-9212

TABLE III: FREQUENCY OF 0'S AND 1'S IN THE GENERATED SEQUENCES IN THE SIX SAMPLES

Sample no	Sequence 1 Frequency of 0's/1's	Sequence 2 Frequency of 0's/1's	Sequence 3 Frequency of 0's/1's	Difference between 1st & 2nd	Difference between 1st & 3rd	Difference between 2nd & 3rd
1	38/26	37/27	30/34	29	30	31
2	32/32	27/37	30/34	33	36	23
3	42/22	29/35	34/30	31	30	33
4	28/36	24/40	31/33	30	39	35
5	29/35	31/33	36/28	32	37	33
6	30/34	32/32	37/27	36	27	31

TABLE IV: FREQUENCY TEST OF THE GENERATED SEQUENCES IN THE SIX SAMPLES

Sample no	Sequence 1 Frequency test	Sequence 2 Frequency test	Sequence 3 Frequency test
1	2.25	1.5625	0.25
2	0	1.5625	0.25
3	6.25	0.5625	0.25
4	1	4	0.0625
5	0.5625	0.0625	1
6	0.25	0	1.5625

TABLE V: SERIAL TEST OF THE GENERATED SEQUENCES IN THE SIX SAMPLES

Sample no	Sequence 1 Serial test	Sequence 2 Serial test	Sequence 3 Serial test
1	1.13	-0.34	-1.69
2	-1.8	-0.34	-0.043
3	4.87	-1.37	-1.31
4	-1.30	2.3	-1.88
5	-0.86	-1.76	-1.3
6	0.21	-0.68	-0.21

TABLE VI: LONGEST RUN OF 0'S AND 1'S IN THE GENERATED SEQUENCES IN THE SIX SAMPLES

Sample no	Longest 0 run in first seq.	Longest 1 run in first seq.	Longest 0 run in second seq.	Longest 1 run in second seq.	Longest 0 run in third seq.	Longest 1 run in third seq.
1	6	5	7	5	6	6
2	4	5	3	5	6	9
3	6	3	4	5	4	5
4	6	5	4	5	6	5
5	4	14	4	5	6	9
6	4	6	9	9	7	3

TABLE VII: INFORMATION ENTROPY TEST FOR THE GENERATED SEQUENCES IN THE 6 SAMPLES

Sample no	Sequence 1	Sequence 2	Sequence 3
1	0.9745	0.9823	0.9972
2	1.0000	0.9823	0.9972
3	0.9283	0.9936	0.9972
4	0.9887	0.9544	0.9993
5	0.9936	0.9993	0.9887
6	0.9972	1.0000	0.9823

VI. CONCLUSION

A block cipher-based pseudo random generator is presented. It is based on a re-keying approach and used the rounds of the RC5 which is a highly random block cipher algorithm. The proposed system generates a large number of sequences in merely a second; the speed of execution is around 200000

Received 13 April 2017; Accepted 3 October 2017

ISSN 1811-9212

sequence/sec. The randomness test shows a good degree of randomness and security and the degree of randomness is still preserved over a long run of sequence generation. The proposed system can be modified to fit the needs of any application.

REFERENCES

[1] Lars R. Knudsen and Matthew Robshaw, The Block Cipher Companion, Springer, 2013.

[2] William Stallings, Cryptography and Network Security: Principles and Practice 6th edition, Pearson, 2014.

[3] László Babai, Trading group theory for randomness, STOC '85 Proceedings of the seventeenth annual ACM symposium on Theory of computing, Pages 421-429,1985.

[4] Ehab Al-Shaer, Xinming Ou and Geoffrey Xie, Automated Security Management, 2013, Springer.

[5] G.K Savvidy, N.G Ter-Arutyunyan-Savvidy, On the Monte Carlo simulation of physical systems, Journal of Computational Physics, 1991.

[6] Matsumoto, Makoto and Nishirmura, Takuji, Mersenne Twister: A 623- dimensionally equidistributed uniform pseudo random number generator, ACM Transactions on Modeling and Computer Simulation, ol. 8, No.1, pp.3-30,1998.

[7] Lagarias J. C. "Pseudorandom Number Generators", Cryptology and Computational Number Theory, edited by C. Pomerance, Proceedings of Symposia in Applied Mathematics 42, American Mathematical Society,1990.

[8] David DiCarlo, Random Number Generation: Types and Techniques, Liberty University, 2012.

[9] Moni Naor and Omer Reingold, Number-theoretic constructions of efficient pseudo-random functions, Journal of the Association for Computing Machinery, 2004.

[10] Ghusn Alban Ali Ahmed, Using The Division Algorithm to Generate Pseudo Random Decimal Sequences, Iraqi Academic Scientific Journals, 2008.

[11] Petit, C., Standaert, F.-X., Pereira, O., Malkin, T., Yung, M.: A block cipher based pseudo-random number generator secure against side-channel key recovery. In: ASIACCS, pp. 56–65, 2008.

[12] Ashwaq T. Hashim and Loay E. George, "Secret Image Sharing Based on Wavelet Transform", International Conference on Information Technology in Signal and Image Processing**,** Mumbai, India, Pp. 324-332, 2014.

[13] Alaa kadhim and Hussein Abed, A New Random Keys Generator Depend on Multi Techniques, Eng. & Tech. Journal, Vol.33,Part (B), No.3, 2015.

[14] Ronald Linn Rivest, "The RC5 Encryption Algorithm". Workshop on Fast Software Encryption (FSE), pp. 86–96, 1994.

[15] AndrewRukhin, JuanSoto, JamesNechvatal, Miles Smid, ElaineBarker, Stefan Leigh, MarkLevenson, Mark Vangel, DavidBanks, AlanHeckert, JamesDray and SanVoB, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22, 2010.

[16] Claude Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, 1948.

[17] Horst Feistel, "Cryptography and Computer Privacy", Scientific American, 1973.

Received 13 April 2017; Accepted 3 October 2017

 ISSN 1811-9212