ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

A Cryptographically Secure Multi-stage Pseudo-random Number Generator

Adi A. Maaita[1], **Hamza A. A. Al_Sewadi** [2], **Abdulameer K. Husain**[3], **Osama Al-Haj Hassan**[4]

Department of Software Engineering, Faculty of IT, Isra University, Amman, Jordan [1,2]

Department of Computer Science, Faculty of IT, Jarash University, Amman, Jordan [3]

Department of Computer Science, Faculty of IT, Isra University Amman, Jordan [4]

Abstract: Random and Pseudo-random number generators are of great interest and are still sought for the benefit of stronger keys for cryptosystems. A new cryptographically secure algorithm for pseudo-random number generation is proposed. The proposed algorithm is characterized by a multi-stage modular design. It combines various processes such as bit swapping, modular operations and secret splitting techniques in order to achieve adequate confusion and diffusion. The obtained results through applying the National Institute of Technology randomness tests clearly show that the algorithm is capable of producing high quality cryptographically secure pseudo-random numbers.

Keywords: Computer security, Cryptosystems, Key distribution, Modular design, Pseudo-random numbers generators, Security agreement.

I. Introduction

Symmetric key encryption algorithms are major components in the security of the digital world. Many symmetric algorithms exist nowadays and many more emerge every day. However, only algorithms that prove to be the most secure find their way to real world applications.

A symmetric key encryption algorithm – as the name implies – uses the same key for both encryption and decryption of a specific piece of data [1]. The choice of the symmetric key is a very important factor to the strength of the encryption process. Choosing a weak key – which is a key that is too easy to guess – is a major encryption related security hazard. This problem is the result of the human inability to remember long and unrelated sequences of symbols. Therefore, humans tend to choose passwords that relate to aspects of their lives, such as birthdays, phone numbers, and names of loved ones.

The only guaranteed way of breaking an encryption is through trying all possible combinations of the symbol set from which the password is composed. This approach is known as the brute-force approach [1]. However, although this approach is guaranteed, it is still a very time consuming process making it infeasible in most situations. Moreover, when the password is composed of symbols relating to aspects of its owner's life, the search space is greatly reduced, and the brute-force approach can be refined to try a limited number of combinations assuming that an attacker has enough information about the victim. Such personal information is gathered through what is known as Social Engineering [2].

Computers on the other hand, do not suffer from the same difficulty in remembering long sequences of unrelated symbols. For that reason, when the encrypting party is a computer program, generated keys can be either truly random or pseudo-random. Truly random numbers (RNs) or keys are unpredictable and can be generated by measuring random physical or natural phenomena such as radioactive decay, atmospheric noise, amplified noise generated by a resistor or a semi-conductor diode, fed to a comparator or Schmitt trigger and then the output is sampled to get a series of bits which are statistically independent or random, and then feeding that seed data to a computer program which generates a random value [3]. Pseudo-random numbers (PRNs) or keys can also be generated by feeding a seed value to a computer program, however, the seed is obtained from a source that has a limited number of seed values, and thus the key exhibits characteristics that are very similar to those of random keys, yet the key is not truly random [4].

Pseudo-random number generators (PRNGs) are usually preferred to true random number generators due to their speed of operation and for the reproducibility of their numbers from an initial value [5]. However, many PRNGs fail to produce bit sequences that exhibit random characteristics, which can lead to undesirable consequences. In order to approve a PRNG as a means for producing pseudo-random bit sequences, bit sequence randomness testing can be applied. A very popular set of pseudo-randomness tests was presented by the National Institute of Standards and Technology (NIST) [6]. These tests statistically examine a bit sequence for randomness characteristics and provide a judgment of whether the bit sequence is random or not. NIST also presented a number of recommendations for PRN generation using deterministic random bit generators [7].

After the brief definitions in Section 1, previous work is summarized in Section 2. Section 3 defines some selected NIST randomness tests which were used to validate the generated pseudo-random bit sequences. The methodology of the proposed pseudo-random number generation algorithm is considered in Section 4. Then section 5 lists and discusses the experimentation and results, and finally Section 6 concludes the work.

II. Literature review

A huge amount of work has been published on the generation of RNs and PRNs as integers, integer sequence, Gaussian, set of decimal fractions, integers that fit normal distribution or numbers in the 0 and 1 range with configurable decimal places. In the following a brief but chronologically arranged work on RNGs is listed.

A decimal procedure for generating RNs designed for decimal machines such the UNIVAC was proposed by Moshman [8]. It followed a suggestion of Von Neumann where an *n* digit number is squared and the middle *n* digits of the resultant number are used as a random number. That number is in turn squared and the process is repeated as many times as desirable.

A 48 bit PRNG was presented by Kuehn [9]. It demonstrated its adequacy for use in Monte Carlo calculations. Pike and Hill [10, 11] suggested a multiplicative congruential method for pseudo-random generation. Brent [12] presented a Gaussian PRNG for the FORTRAN language. The presented algorithm generates numbers which are independent and normally distributed on [0, 1).

Impagliazzo and Levin [13] and Hastad [14] suggested using one-way functions for pseudo-random number generation. Deng et al. [15] suggested a number of methods for enhancing the yield of pseudo-random number generators, thus producing sequences which are asymptotically independent and uniformly distributed.

Bellare et al. [16] presented the concept of distributed pseudo-random number generators. Matsumoto and Nishimura [17] presented an algorithm which they named the Mersenne Twister (MT). The proposed algorithm had a longer period and larger k-distributions when compared to existing pseudo-random number generators at the time. Ren [18] presented a PRNG that is capable of generating sequences of long period, large complexity, balanced statistics, and low cross-correlation from the addition of *m*-sequences with pairwise-prime linear spans (AMPLS).

Xuelong et al. [19] proposed the use of one-dimensional extended non-uniform cellular automata to generate pseudo-random bit sequences. The cellular automata rules were generated using a genetic algorithm.

Petit et al. [20] proposed a design for pseudo-random number generation based on the block cipher design. The proposed design showed very good resistance against side-channel attack strategies.

Using of graphics hardware for generating PRNs was proposed by Langdon [21]. He presented a high speed pseudo-random number generator for the NVidia Cuda parallel architecture using C++ as the implementation language.

PRNGs based on chaotic iterations were also suggested by Bahi et al. [22-24], for watermarking applications. They proposed chaotic dynamical systems which appear to be good candidates to achieving a mix of secure and fast PRNGs in order to benefit from their respective qualities.

Mitra and Kundu [25] proposed a cost effective design methodology for pseudo-random generation using cellular automata and they presented an Equal Length Cellular Automata based PRNG.

Heike et al. [26] suggested the generation of PRNs by applying iteration to a one-way function, based on an initial value and a random key to generate part of the PRN that is used as a key in the next iteration step of the one-way function.

Elsherbeny et al. [27] proposed a new Deterministic Chaotic System that implements an Iteration Function System (IFS) for generating a PRN. In this system and at a certain initial value, the iterated function generates chaotic random numbers.

Splittable pseudo-random number generators (PRNGs) were very useful for structuring purely functional programs that deal with randomness, because they allow different parts of the program to independently generate random values, thus avoiding random seed threading through the whole program. Claessen and Palka [28] proposed a Splittable PRNG using a cryptographic hash function. In this scheme, the authors showed that the currently known and used splittable PRNGs are either not efficient enough, have inherent flaws, or lack formal arguments about their randomness. They provided proofs of randomness guarantee under some cryptographic assumptions.

This paper suggests a PRNG which encompasses a multi-stage algorithm that implements bitwise manipulation. It is designed to achieve adequate bit string confusion and diffusion by combining various processes such as bit swapping, modular operations and secret splitting techniques. Experimental results show that the presented PRNG is cryptographically secure through the application of the NIST randomness tests.

III. Randomness Tests

PRNGs are either secure but slow, or fast but insecure [29]. In addition, they are either not efficient enough, have inherent flaws, or lack formal arguments for their randomness. Claessen and Palka [28] provided proofs in order to show strong guarantees of randomness under assumptions commonly made in cryptography. Also, NIST [6] published a set of statistical tests; six of them will be used for validation purposes here. These tests are the Frequency (Monobit) test, Frequency within a Block test, the Runs test, the Longest-Run-of-Ones in a Block test, the Binary Matrix Rank test, and the Discrete Fourier Transform (or Spectral) test. The most important value to calculate in these tests is the *P-value*, which is to be compared with a significant probability level α. For cryptographic applications, α is found to have a value of about 0.01. α is defined as the probability that a randomness test of the generated number indicates that it is not random when it is really random. Moreover, the *P-value* is the probability that a perfect random number generator would have produced a bit sequence less random than the sequence that was tested. The testing criteria applied is if *P-value* $>= \alpha$, then the sequence appears to be random but if *P-value* $< \alpha$, then the sequence appears to be non-random. A short summary of the selected randomness tests is given below.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

A. *Frequency (Monobit) Test*

It tests the proportion of 0's and 1's for the entire sequence in order to determine whether they are approximately the same as would be expected for a truly random sequence. Given a number $\varepsilon = b_1, b_2 ..., b_n$, then the observed value S_{obs} is calculated and used determine the *P-value* as defined by eq 1.

$$S_{obs} = \frac{|S_n|}{\sqrt{n}} \text{, then } P-value = erfc\left\{\frac{S_{obs}}{\sqrt{n}}\right\} \ . . . (1)$$

Where S_n is the sum of all string bits after converting each 0 to -1, and *erfc* is the complementary error function.

B. *Frequency Test within a Block*

This test determines if the frequency of 1's in an M-bit blocks is approximately *M/2*, as expected under an assumption of randomness. The *P-value* is calculated by eq 2.

$$P-value = igamc\left(N/2, \chi^2(obs)/2\right) \ . . . (2)$$

Where *igamc* is the incomplete gamma function, *N* is the number of M-bit blocks to be tested, and $\chi^2(obs)$ is the chi function of the observed proportion of 1's within a given M-bit block.

C. *The Runs Test*

It tests the total number of uninterrupted sequence of identical bits; *i.e.* it indicates the speed of 1's and 0's whether it is too fast or too slow. The *P-value* is calculated by eq 3.

$$P-value = erfc\left\{\frac{|V_n(obs) - 2n\pi(1-\pi)|}{1\sqrt{2n}\pi(1-\pi)}\right\} \ . . . (3)$$

Where $V_n(obs)$ is the total number of run across n and π is the pre-test proportion in the input sequence.

D. *Longest-Run-of-Ones in a Block Test*

It tests the longest run of 1's within M-bit blocks, and its consistency with that expected in a random sequence. The P-value is calculated by eq 4.

$$P-value = \left\{\frac{K}{2}, \frac{\chi^2(obs)}{2}\right\} \ . . . (4)$$

Where $\chi^2(obs)$ is a matching measure between observed longest run length within M-bit blocks with the expected longest length within M-bit blocks, *i.e.* it gives how well the observed number of ranks of various orders match the expected number of ranks under an assumption of randomness.

E. *Matrix Rank*

This test checks for linear dependence among fixed length substrings of the original sequence. The *P-value* is calculated by eq 5.

$$P - value = e^{-X^2(obs)/2} . . . (5)$$

Where $\chi^2(obs)$ is a measure of how well the observed number of ranks of various orders match the expected number of ranks under an assumption of randomness.

F. *Discrete Fourier Transform (or Spectral) Test*

It tests the peak heights in the Discrete Fourier Transform of the sequence in order to detect periodic features in the sequence that indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95 % threshold is significantly different from 5 %. The *P-value* is calculated by eq 6

$$. \quad P - value = erfc(\frac{|d|}{\sqrt{2}}) \ . . . (6)$$

Where d is the normalized difference between the observed and the expected number of frequency components that are beyond the 95 % threshold.

IV. The Proposed PRNG Methodology

The PRNG algorithm proposed in this paper starts with an agreed upon seed that could be s used as a key for a cryptographic system. This seed is randomly selected and may consist of any combination of letters (lower or upper case) and numbers. The seed components are replaced by their ASCII code binary representation as they enter to the PRNG. This algorithm can generate PRN of any length; however, for the purpose of assuring randomness by NIST [6] standard tests, the selected seed shall be of length *n*128* bits where n is an integer value larger than 1.

Shannon's principle of diffusion and confusion [30] was achieved in the algorithm design through conducting three successive stages covering bit manipulation such as bit swapping and modular operations, logical operations, and finally a secret splitting technique as detailed below.

A. *Stage 1:*

This stage achieves bits string confusion by initially swapping the bits of the original seed and then applying modular operations. It consists of the following 4 steps as illustrated in fig 1.

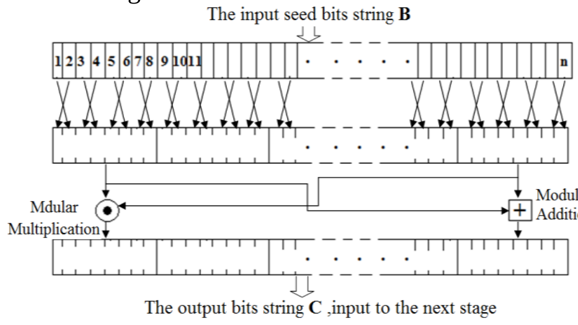
The input seed bits string **B**
1 2 3 4 5 6 7 8 9 10 11
n
Mdular Multiplication
Modular Addition
The output bits string **C** ,input to the next stage

Fig 1. Work flow for stage 1.

Step 1: On receiving the seed in binary form, bit string confusion is achieved by interchanging or swapping adjacent bits. Therefore if the seed bits string is "$b_1b_2b_3b_4...b_n$", then the resulting bit string by eq. 7.

$$\mathbf{B} = "\mathbf{b_1 b_2 b_3 b_4 ... b_n}" \quad . . . . (7)$$

Step 2: The resulting number B, is split into blocks of 32 bits length each, eq. 8.

$$\mathbf{B} = "\mathbf{B_1, B_2, B_3, ..., B_k}" \text{, where } k = n/32 \ . . . (8)$$

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

Step 3: The far most blocks (B_1& B_k) are treated two modular operations. This step performs modular multiplication, resulting into blocks C_1, and then the next far most blocks (B_2& B_{k-1}), are multiplied resulting into blocks C_2 and so on, giving the results by eq 9.

$$\mathbf{C_i} = (\mathbf{B_i} \bullet \mathbf{B_j}) \bmod 2^{32} + 1 \quad . . . . (9)$$

Where i = 1, 2, 3, …, k/2 and j = k, k-1, k-2, …, (k/2) +1

Step 4: Similarly as in step 3, the same blocks are treated by performing modular addition instead of modular multiplication and the results are given by eq. 10.

$$\mathbf{C_j} = (\mathbf{B_i} + \mathbf{B_j}) \bmod 2^{32} \quad . . . . (10)$$

The resulting bit string of this stage will be as in eq 11.

$$\mathbf{C} = "\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3, \ldots, \mathbf{C_k}" \quad . . . . (11)$$

B. Stage 2:

The obtained bit string C of the previous stage is used as input to the next stage. Bitwise XOR operations are performed on its components successively as illustrated in fig 2. Each bit is XORed with that next to it from left to right, while the least significant bit is finally XORed with the most significant bit. The resulting bit string is given by eq. 12.

$$\mathbf{D} = "\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3, \ldots, \mathbf{D_n}" \quad . . . . (12)$$

Where, $\mathbf{D_i} = \mathbf{C_i} \otimes \mathbf{C_{i+1}}$, for *i* = *1* *to* *n-1* and $\mathbf{D_n} = \mathbf{C_n} \otimes \mathbf{C}_1$.

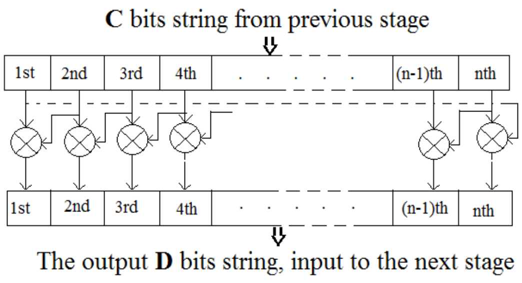
C bits string from previous stage
1st
2nd
3rd
4th
(n-1)th
nth
1st
2nd
3rd
4th
(n-1)th
nth
The output D bits string, input to the next stage

Fig 2. Work flow for stage 2.

C. Stage 3:

In order to introduce more control on the generated PRN, some information must be agreed upon by the users in advance. This would enable the users to generate their own random numbers to be used for cryptographic systems. The adopted structure of this agreed upon information in the proposed algorithm consists of a predefined procedure to group the obtained bit string *D* of stage 2 into blocks of specific length (8 bits here), regroup theses blocks according to a specified different secret splitting weight for each segment, manipulate with these segments as shown in the following steps.

Step1: split *D* into *k* segments of certain length, and each segment is considered to be one cell. Therefore, the number of cells *k* = *n / segment bits* (*k* = *n / 8* here).

Step 2: Convert each segment into bytes and place that segment in the cell.

Step 3: Use the agreed upon splitting weights or percentages, such as r_1 = 25%, r_2 = 35%, and r_3 = 45%. It must be noted that any number of weights can be used.

Step 4: These cells are decomposed through a successive segmentation process based on the given splitting weights multiplied by the remaining number of cells in each run, which is achieved by taking the integer part only of the result and then use it as the number of cells in each new segment. This step is demonstrated as follows:

$R_1 = \llcorner\ k * r_1\% \lrcorner$,
$R_2 = \llcorner\ (k - R_1) * r_2\% \lrcorner$,
$R_3 = \llcorner\ (k - (R_1+R_2)) * r_3\% \lrcorner$,
$R_4 = \llcorner\ (k - (R_1+R_2+R_3)) * r_1\% \lrcorner$,
And so on until all k cells are considered.

Step 5: Each new segment is then XORed with its corresponding weight.

Step 6: Convert the output of step 5 back to binary number.

At the end of this step, the system generates the first set of pseudo-random numbers.

Step 7: In order to generate the next set of random numbers, the obtained PRN of step 6 is fed as input to stage 1, then all the steps of stages 1, 2, and 3 are repeated.

V. EXPERIMENTATION AND RESULTS

A software was written in C# language, implementing the NIST tests utilised for the experiments conducted on the proposed PRNG algorithm. The software is designed to accept a seed of any number of characters and generate as many random numbers as practically required. The used seeds in these experiments are varying length binary sequences obtained from the binary expansion of e (exponent). This choice of seed was made in order to make the experiments easily repeatable. Experiments were carried out for the generation and testing of random keys of 128, 512, 1024, and 2048 bit lengths. However, these tests can be carried out practically for any key length. All the generated numbers were tested for randomness through the six elaborate tests outlined in section 3, namely frequency test, frequency block test, runs test, longest run test, binary matrix test and discrete Fourier transform (spectral) test.

The potential problems with PRNGs are their failure in statistical pattern-detection tests. The causes of failures are usually attributed to many reasons such as lack of distribution uniformity; correlation of successive values, and the output sequence has poor dimensional distribution and a shorter seed state.

In order to see the progress of randomness generation throughout the three stages of the algorithm, the PRNG algorithm is run using the secret splitting weights r1=20%, r2=30%, and r3=50%. The obtained success rates for random sequence generation for key lengths 128, 256, 512, 1024, and 2048 bits is listed in tables I - III.

TABLE I. FREQUENCY TEST AND FREQUENCY BLOCK TEST RESULTS FOR THE THREE STAGES.

Key length	Frequency			Frequency Block		
	Stage 1	Stage 2	Stage 3	Stage 1	Stage 2	Stage 3
128 bit	99.1%	98.9%	98.3%	99.4%	99.4%	99.2%
256 bit	98.8%	98.5%	99.3%	99.1%	98.8%	98.7%
512 bit	99.2%	98.9%	99.3%	98.9%	98.5%	98.6%
1024 bit	99.2%	98.8%	99.1%	99.4%	99.2%	99.8%
2048 bit	98.3%	98.5%	99.1%	99.3%	98.7%	99.3%

TABLE II. RUNS TEST AND LONGEST RUN TEST RESULTS FOR THE THREE STAGES

Key length	Runs test			Longest Run test		
	Stage1	Stage2	Stage3	Stage1	Stage2	Stage3
128 bit	98.5%	99.2%	99.7%	99.0%	98.2%	99.1%
256 bit	98.4%	99.3%	99.2%	99.0%	99.0%	99.1%
512 bit	98.9%	99.1%	98.9%	98.9%	98.3%	99.1%
1024 bit	99.1%	98.9%	98.9%	98.8%	99.4%	99.3%
2048 bit	98.4%	98.9%	99.1%	98.5%	98.8%	98.5%

TABLE III. BINARY MATRIX TEST AND DISCRETE SPECTRAL TEST RESULTS FOR THE THREE STAGES.

Key length	Binary Matrix			Discrete Spectral test		
	Stage1	Stage2	Stage3	Stage1	Stage2	Stage3
128 bit	99.0%	99.0%	98.9%	98.1%	98.2%	98.4%
256 bit	98.5%	98.6%	99.1%	98.8%	98.5%	98.7%
512 bit	99.1%	99.2%	98.8%	98.8%	98.9%	98.9%
1024 bit	99.0%	98.6%	99.1%	98.7%	98.3%	99.1%
2048 bit	98.4%	98.1%	98.7%	98.4%	98.8%	99.0%

The results of the six tests for the three stages show the improvement in randomness along the stages which justifies the use of more than one stage in the proposed algorithm.

The tests for the generated pseudo-random sequences by the algorithm using all the six tests are listed in table IV. This table lists the rate of success for key lengths of 128, 256, 512, 1024, and 2048 bits. The calculated rates of success of the generated PRNs varies between 98.4% and 99.1% for the tests used, therefore, the proposed algorithm is producing satisfactory randomness. This randomness is confirmed by fig 3, which shows the algorithm rate of failure as a function of the number of iterations. All the six tests involved manifested convergence in the rate of failure to a very small value around 1% as the number of iterations increase beyond 2000 epoch.

TABLE IV. PERCENTAGES OF BIT SEQUENCES PASSING THE NIST TESTS FOR THE PROPOSED PRNG ALGORITHM

(FOR SECRET SPLITTING PERCENTAGES; R1=20%,

Key Length (bits)	Frequency	Frequency Block	Runs	Longest Run	Binary Matrix	DST
128	98.3%	99.2%	99.7%	99.1%	98.9%	98.4%
256	99.3%	98.7%	99.2%	99.1%	99.1%	98.7%
512	99.3%	98.6%	98.9%	99.1%	98.8%	98.9%
1024	99.1%	99.8%	98.9%	99.3%	99.1%	99.1%
2048	99.1%	99.3%	99.1%	98.5%	98.7%	99.0%

R2=30%, AND R3=50%)

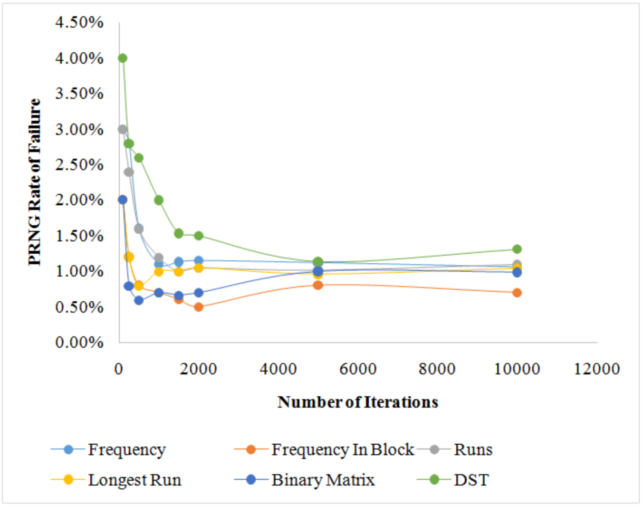

Fig 3. The rate of failure for different tests along the number of iterations.

An investigation is also conducted seeking the best combination of secret splitting percentages that produce better pseudo-random number generation. However, no significant differences were noticed in the calculated success rates for various secret splitting percentages in the case of all the key sequence lengths involved, i.e. 128, 256, 512, 1024, and 2048 bits.

However, it was noticed that the 1024 bits key sequence length has given slightly better randomness success rate than others considered. The results of running the PRNG algorithm for this case are listed in table V, and illustrated in fig 4.

TABLE V: THE AVERAGE RATE OF SUCCESS FOR DIFFERENT TESTS ON SEQUENCES USING DIFFERENT PERCENTAGE VALUES OVER 10000 ITERATIONS

Key Length (bits)	Frequency	Frequency Block	Runs	Longest Run	Binary Matrix	DST
128	98.3%	99.2%	99.7%	99.1%	98.9%	98.4%
256	99.3%	98.7%	99.2%	99.1%	99.1%	98.7%
512	99.3%	98.6%	98.9%	99.1%	98.8%	98.9%
1024	99.1%	99.8%	98.9%	99.3%	99.1%	99.1%
2048	99.1%	99.3%	99.1%	98.5%	98.7%	99.0%

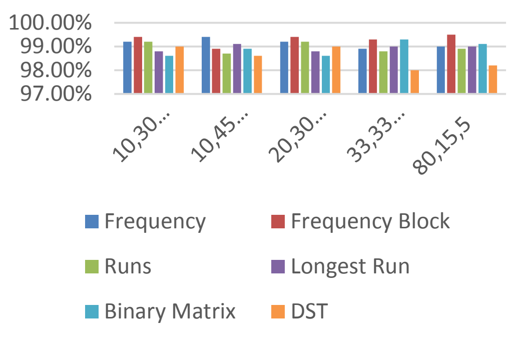

Fig 4. The average rate of success for different tests on sequences using different percentage values over 10000 iterations.

VI. SECURITY ISSUES

Mixing of bitwise manipulation or swapping, performing modular operations (multiplication and addition), implementing Boolean operations (XOR), and secret splitting processes according to predefined weights serves to avoid purely algebraic attacks and the purely bit oriented attacks. They also prevent the mathematical behaviour of the scheme from being shaped easily. They both contribute to a great increase in mathematical complexity together with high computational efficiency.

Moreover, all the adopted operations can be easily and efficiently implemented whether by hardware or by software.

VII. CONCLUSION

An algorithm for computationally fast, cryptographically secure pseudo-random key generation has been proposed and described in this paper. It is a multi-stage algorithm based on mixing bitwise Boolean operations, integer modular operations, along with bit manipulations and displacements for secret splitting. The experimental part of the paper demonstrated that an average of 98.9% the generated sequences were unpredictable and passed successfully six tests proposed by the NIST.

REFERENCES

[1] Schneier, B., Applied Cryptography: Protocols, Algorithms, and Source Code in C, ed. S. Phil. 1995: John Wiley \\&amp; Sons, Inc. 758.
[2] Krombholz, K., et al., Social engineering attacks on the knowledge worker, in Proceedings of the 6th International Conference on Security of Information and Networks. 2013, ACM: Aksaray, Turkey. p. 28-35.
[3] L. Larger, a.J.M.D., Nonlinear dynamics Optoelectronic chaos. Nature, 2010. 465(7294): p. 41-42.
[4] Q. Wang, J.B., C. Guyeux, and X. Fang, Randomness quality of CI chaotic generators; application to internet security, in INTERNET'2010. The 2nd International Conference on Evolving Internet. 2010, IEEE Computer Society Press: Valencia, Spain. p. 125–130.
[5] Stockmal, F., Calculations with Pseudo-Random Numbers. J. ACM, 1964. 11(1): p. 41-52.
[6] Lawrence E. Bassham, I., et al., SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. 2010, National Institute of Standards \& Technology.
[7] Barker, E.B. and J.M. Kelsey, SP 800-90A. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. 2012, National Institute of Standards \& Technology.
[8] Moshman, J., The Generation of Pseudo-Random Numbers on a Decimal Calculator. J. ACM, 1954. 1(2): p. 88-91.
[9] Kuehn, H.G., A 48-bit pseudo-random generator. Commun. ACM, 1961. 4(8): p. 350-352.
[10] Pike, M.C. and I.D. Hill, Algorithm 266: pseudo-random numbers [G5]. Commun. ACM, 1965. 8(10): p. 605-606.
[11] Pike, M.C. and I.D. Hill, Remark on algorithm 266: pseudo-random numbers. Commun. ACM, 1966. 9(9): p. 687.
[12] Brent, R.P., Algorithm 488: A Gaussian pseudo-random number generator. Commun. ACM, 1974. 17(12): p. 704-706.
[13] Impagliazzo, R., L.A. Levin, and M. Luby, Pseudo-random generation from one-way functions, in Proceedings of the twenty-first annual ACM symposium on Theory of computing. 1989, ACM: Seattle, Washington, USA. p. 12-24.
[14] Håstad, J., Pseudo-random generators under uniform assumptions, in Proceedings of the twenty-second annual ACM symposium on Theory of computing. 1990, ACM: Baltimore, Maryland, USA. p. 395-404.
[15] Deng, L.-Y., E.O. George, and Y.-C. Chu, On improving pseudo-random number generators, in Proceedings of the 23rd conference on Winter simulation. 1991, IEEE Computer Society: Phoenix, Arizona, USA. p. 1035-1042.
[16] Bellare, M., J.A. Garay, and T. Rabin, Distributed pseudo-random bit generators—a new way to speed-up shared coin tossing, in Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing. 1996, ACM: Philadelphia, Pennsylvania, USA. p. 191-200.
[17] Matsumoto, M. and T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. 1998. 8(1): p. 3-30.
[18] Ren, J., Design of long period pseudo-random sequences from the addition of m-sequences over Fp. EURASIP J. Wirel. Commun. Netw. 2004. 2004(1): p. 12-18.
[19] Xuelong, Z., et al., High-quality pseudo-random sequence generator based on one-dimensional extended cellular automata, in Proceedings of the 3rd international conference on Information security. 2004, ACM: Shanghai, China. p. 222-223.
[20] Petit, C., et al., A block cipher based pseudo random number generator secure against side-channel key recovery, in Proceedings of the 2008 ACM symposium on Information, computer and communications security. 2008, ACM: Tokyo, Japan. p. 56-65.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

[21] Langdon, W.B., A fast high quality pseudo random number generator for nVidia CUDA, in Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. 2009, ACM: Montreal, Québec, Canada. p. 2511-2514.
[22] J. Bahi, C.G., and Q. Wang, A novel pseudo-random generator based on discrete chaotic iterations, in INTERNET'09, 1-st International conference on Evolving Internet. 2009: Cannes, France. p. 71–76.
[23] J. Bahi, C.G., and Qianxue Wang, A pseudo random numbers generator based on chaotic iterations; Application to watermarking, in International conference on Web Information Systems and Mining, WISM 2010. 2010: Sanya, China. p. 202–211.
[24] J. M. Bahi, a.C.G., Topological chaos and chaotic iterations, application to hash functions, in IEEE World Congress on Computational Intelligence WCCI. 2010: Barcelona, Spain. p. 1–7.
[25] Mitra, A. and A. Kundu, CA based cost optimized PRNG for Monte-Carlo simulation of distributed computation, in Proceedings of the CUBE International Information Technology Conference. 2012, ACM: Pune, India. p. 332-337.
[26] Heike B. Neumann, S.S., Matthias Voegeler, Method of generating pseudo-random numbers. 2009.
[27] Mohamed Nageb Elsherbeny, M.R., Pseudo –Random Number Generator Using Deterministic Chaotic System. INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, 2012. 1(9).
[28] Claessen, K.P., M, Splittable Pseudorandom Number Generators using Cryptographic Hashing, in Proceedings of Haskell Symposium. 2013. p. 47-58.
[29] H. B. Neumann, S.S., and M. Voegeler, Method of generating pseudo-random numbers. 2009.
[30] Shannon, C.E., Communication Theory of Secrecy Systems. Bell System Technical Journal, 1949. 28(4): p. 656-715.

BIOGRAPHIES

Adi A. Maaita is currently an assistant professor at the Faculty of Information Technology, Isra University. He received his B.Sc. degree in 2002 from the University of Jordan (Jordan), his M.Sc.in 2003 from the New York Institute of Technology (Jordan).Then received his Ph.D. from the University of Leicester (UK) in 2008. His research interests include Cryptography, Steganography, Information and Computer Network Security, Genetic Algorithms, Neural Networks, and Software Modelling.

Hamza A. A. Al_Sewadi is currently a professor at the Faculty of Information Technology, Isra University (Jordan). He got his B.Sc. degree in 1968 from Basrah University, Iraq, then M.Sc. and Ph.D. degrees in 1973 and 1977 respectively, from University of London (UK). He worked as associate professor at various universities such as Basrah University (Iraq), Zarqa University and Isra University (Jordan), visiting professor at University of Aizu (Japan). His research interests include Cryptography, Steganography, Information and Computer Network Security, Artificial Intelligence and Neural Networks.