International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:4, 2015

Deterministic Random Number Generator Algorithm for Cryptosystem Keys

Adi A. Maaita, Hamza A. A. Al_Sewadi

***Abstract*—**One of the crucial parameters of digital cryptographic systems is the selection of the keys used and their distribution. The randomness of the keys has a strong impact on the system's security strength being difficult to be predicted, guessed, reproduced, or discovered by a cryptanalyst. Therefore, adequate key randomness generation is still sought for the benefit of stronger cryptosystems. This paper suggests an algorithm designed to generate and test pseudo random number sequences intended for cryptographic applications. This algorithm is based on mathematically manipulating a publically agreed upon information between sender and receiver over a public channel. This information is used as a seed for performing some mathematical functions in order to generate a sequence of pseudorandom numbers that will be used for encryption/decryption purposes. This manipulation involves permutations and substitutions that fulfill Shannon's principle of "confusion and diffusion". ASCII code characters were utilized in the generation process instead of using bit strings initially, which adds more flexibility in testing different seed values. Finally, the obtained results would indicate sound difficulty of guessing keys by attackers.

***Keywords*—**Cryptosystems, Information Security agreement, Key distribution, Random numbers.

I. Introduction

INDEPENDENT, unpredictable and uniformly distributed numbers that cannot be reliably reproduced are referred to as random numbers [1]. They play a major part in practical implementation and strength of most cryptographic systems. They may be used as keys for symmetric crypto-systems, public key parameters, session keys, etc. [2]. Failure of obtaining strong keys definitely will end up with data security compromise. Therefore, strong random number generators that exhibit high statistical quality and can withstand cryptanalysis efforts are keenly sought. Such strong random number generators constitute an important building block in the design and testing of high quality crypto-systems [3].

Generally random numbers can be truly random TRN, pseudo-random PRN or quasi-random QRN. Truly random numbers are unpredictable. Their generation stems from random physical or natural phenomena, such as radioactive decay, 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 [4].

Adi A. Maaita is with the Faculty of Information Technology, Isra University, Amman, Jordan (phone: 00962798144676; e-mail: adi.maaita@iu.edu.jo,

Hamza A. A. Al_Sewadi is with the Faculty of Information Technology, Isra University, Amman, Jordan (phone: 00962795906054, e-mail: alsewadi@hotmail.com).

Pseudo-random numbers generators, also known as deterministic random bit generators, are computer programs that generate a sequence of numbers whose properties approximate the properties of sequences of random numbers [5], [6]. These sequences are not truly random as they are completely determined by a relatively small set of initial values called seeds; however, they are important in practice for their speed and reproducibility in number generation.

Quasi-random numbers are sequences in arbitrary dimensions which progressively cover a d-dimensional space with a set of points that are uniformly distributed. They are also known as low-discrepancy sequences [7]. The quasi-random sequence generators use an interface that is similar to the interface for random number generators, except that seeding is not required as each generator produces a single sequence.

Recently, a new type of generators which are called Lagged Fibonacci pseudo-random number generators [8], [9] have become increasingly popular generators for serial as well as scalable parallel machines. They are proved to be easy to implement, cheap to compute and they are performing reasonably well on standard statistical tests especially when the lag is sufficiently high.

After the brief definitions in Section I, related works will be summarized in Section II. Section III defines the important randomness tests that will be executed to examine the randomness of the generated keys. Section IV explains the proposed Pseudo-random generator scheme; Section V includes the implementation of the proposed scheme and the results of the randomness tests. Finally Section VI concludes the paper.

II. Related Work

Random number generators may be classified into Integer generators, sequence generators, integer set generators, Gaussian generators, decimal fraction generators or row random byte generators depending if they generate integers, integer sequence, set of random integers, integers that fits normal distribution or numbers in the 0 and 1 range with configurable decimal places, respectively. Each of the mentioned types is useful for many cryptographic purposes [10]. Splitable pseudorandom 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 [11].

The available pseudorandom number generators (PRNGs)

are either secure but slow, or fast but insecure [12]. Besides, they are either not efficient enough, have inherent flaws, or lack formal arguments for their randomness. Claessen and Palka [12] provided proofs in order to show strong guarantees of randomness under assumptions commonly made in cryptography.

Mixing secure and fast PRNGs in order to benefit from their respective qualities was sought recently, for example [14] proposed chaotic dynamical systems which appear to be good candidates to achieve this mixture for optimization by topological chaos and chaotic iterations for hash functions, [13]. PRNGs based on chaotic iterations were suggested also for watermarking application [14] and [15].

Chaotic systems have many advantages as unpredictability or disorder-like, which are required in building complex sequences, [16], [17]. This is why chaos has been applied to secure optical communications as suggested by [18]. However, chaotic systems of real-number or infinite bit representation realized in finite computing precision have the problems of non-ideal distribution and short cycle length. Hence infinite space of integers was considered lately leading to the proposition of using chaotic iterations (CIs) techniques. Such proposal has to a new family of statistically perfect and fast PRNGs, [19], where a new version of this family has been proposed. It uses decimation strategies that lead to improvements in both random number generation speed and statistical qualities. Other interesting PRNGs used a new Iteration Function System (IFS) that measures the sensitivity of the IFS to certain initial values in order to generate chaotic random numbers, [20].

PRNGs were also suggested based on iterative implementation of one-way functions utilizing a randomly selected start value with a key, [21]. Both the start value and the key for subsequent iterations were selected from the already generated random number in the previous iteration.

Another interesting PRNG method was suggested by [22]. It introduced a dynamic system to produce an interesting hierarchy of random numbers based on the review of random numbers characteristics and chaotic functions theory. The authors had carried out certain statistical tests on a series of numbers obtained from the introduced hierarchy.

Orue et al. [23] suggested cryptographically secure PRNGs that are based on the combination of the sequences generated by three coupled Lagged Fibonacci generators, mutually perturbed. The mutual perturbation method consists of the bitwise XOR cross-addition of the output of each generator with the right-shifted output of the nearby generator. The proposed generator has better entropy and much longer repetition period than the conventional Lagged Fibonacci Generator.

This paper proposes a pseudo-random number generation scheme that is based on Shannon's concept of confusion and diffusion. The generated random numbers are to be used for cryptographic application. It suggests an in house scenario process that implements a one-time-pad key for secure communication. The scheme generates continuous strings of random bit sequences to be used as one time keys progressively for the subsequent messages.

III. Key Randomness Test

The generated pseudorandom binary sequences can be tested for randomness by some of the statistical tests outlined by NIST [7]. These tests will focus on a variety of different types of non-randomness. The selected tests here include Frequency (Monobit) test, Frequency test within a Block, the Runs test, and the test for the Longest-Run-of-Ones in a Block and will be summarized below. For any of these tests the P-value is calculated and compared against the level of significance α (whose value is commonly set to about 0.01 for cryptographic applications). α is defined as the probability that the generated number is not random when it is really random and P-value is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test [7]. The criteria is if P-value>=α, the sequence appears to be random but if P-value<α, then the sequence appears to be non-random.

A. Frequency (Monobit) Test:

It tests the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

For the purpose of testing the randomness of a number εwith a bit string length of n bits, such that ε= $b_1, b_2 \ldots, b_n$, an observed value S_{obs} is used as a test statistic which is defined by (1).

$$S_{obs} = \frac{|S_n|}{\sqrt{n}} \tag{1}$$

where S_n is the sum of all string bits after converting zeros and ones to -1 and +1, respectively.

The P-value for this test is calculated by (2)

$$P - value = erfc\left\{\frac{S_{obs}}{\sqrt{n}}\right\} \tag{2}$$

where *erfc* is the complementary error function [7]

B. Frequency Test within a Block test:

It tests the proportion of 1's within M-bit blocks. The purpose of this test is to determine whether the frequency of 1's in an M-bit blocks is approximately M/2, as would be expected under an assumption of randomness. The *P-value* is calculated by (3):

$$P\_value = igamc\left(N/2, \chi^2(obs)/2\right) \tag{3}$$

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 given by (4).

$$\chi^2(obs) = 4M \sum_{i=1}^{N} \left(\frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+1}}{M} - \frac{1}{2} \right) \quad (4)$$

C. The Runs Test:

It tests is the total number uninterrupted sequence of identical bits, i.e. whether the number of runs of 1's and 0's of various lengths is as expected for a random sequence. This test indicates the speed of 1's and 0's whether it is too fast or too slow. The *P-value* is calculated by (5):

$$P - value = erfc \left\{ \frac{\left| V_n(obs) - 2n\pi(1-\pi) \right|}{1\sqrt{2n}\pi(1-\pi)} \right\} \quad (5)$$

Where V_n(obs) is the total number of run across n and π is the pre-test proportion in the input sequence given by (6):

$$\pi = \frac{\sum_j \varepsilon_j}{n} \quad (6)$$

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

It tests the longest run of 1's within M-bit blocks, and show whether it is consist with the length of the longest run of 1's that would be expected in a random sequence. The *P-value* is calculated by (7) and $\chi^2(obs)$ is a measure of matching between observed longest run length within M-bit blocks with the expected longest length within M-bit blocks, given by (8):

$$P\_value = \left\{ \frac{K}{2}, \frac{\chi^2(obs)}{2} \right\} \quad (7)$$

$$\chi^2(obs) = \sum_{i=0}^{K} \frac{(\upsilon_i - N\pi_i)^2}{N\pi_i} \quad (8)$$

Where v_i is the frequencies of the longest runs of 1's in each block categorized for i=0 to K while the values of K and N are determined by the value of M in accordance with the pre-set Table I.

TABLE I
PRE-SET VALUES FOR M, K AND N WITHIN THE NUMBER

Minimum key length *n*	*M*	*K*	*N*
128	8	3	16
6272	128	5	49
750 000	10000	6	75

IV. Pseudorandom Number Generation Methodology

This work proposes a scheme for pseudorandom number generator (PRNG) that combines bitwise logical operation and bits manipulation in order to fulfill the confusion and diffusion principle. It starts with a randomly selected input key (seed) that consists of any combination of letters (lower or upper case) and numbers (i.e. a b c … z, A B C … Z , _ , 0 1 2 … 9). This seed can be exchanged between sender and receiver publicly but would even better if it is exchanged secretly. They are replaced by their ASCII code binary representation as they enter to the PRNG. The length of this key is decided by the cryptographic system that is going to be implemented, for example, the 64 bits key for DES needs 8 characters and the 128 bits key for AES needs 16 characters, and so on. The work flow diagram of suggested PRNG scheme is illustrated in Fig. 1.

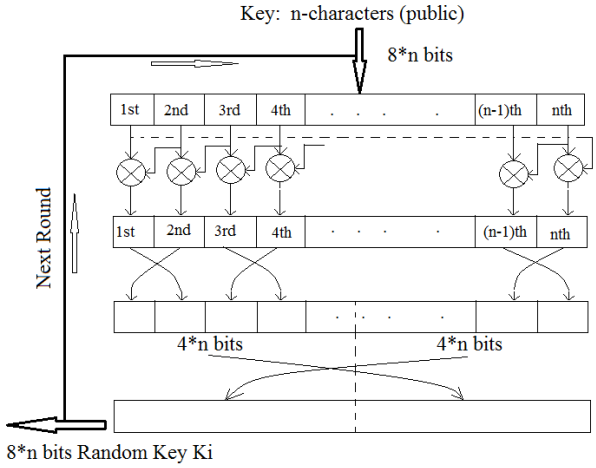

Fig. 1 Work flow diagram for the proposed PRNG

Prior to the operation of the proposed PRNG, the characters of the supplied key are converted to ASCII codes representation, and then the following steps are performed according to the work flow of Fig. 1.

Step 1. The selected key characters are first replaced by their binary representations. Let the entered key consists of n-characters, then the length of this key will be 8*n (i.e. n-bytes).

Step 2. Abitwise XOR operations are performed on the bit blocks of each two successive bytes, i.e. (1^{st}XOR2^{nd}) replaces the 1stbyte, (2^{nd} XOR 3^{rd}) replaces 2nd character, etc., until the last byte where it is XOR'ed with the first one, or (n^{th}XOR1^{st}) replaces the n^{th}-byte, as given by (9):

$$\text{i.e. } i^{th} \, XOR \, (i+1)^{th} \bmod n \text{ , for i=1 to } n \quad (9)$$

Step 3. Successive bytes are exchanged with each other in pairs. However, if n is odd number, then the last byte is left unaltered.

Step 4. The resulting bit string of the previous step is divided into halve, left and right each of 4*n bits length. The resulting bit sequence of 8*n can be taken as the first pseudorandom random key K_1.

Step 5. The generated key in step 4 can be fed back as an input to step 2 in order to generate next random key.

Step 6. In order to generate more keys, steps 2-5 can be repeated as many as required.

A computer program is written to perform these steps written in C# language and tabulate the results in excel sheet together with their randomness tests in order to be ready for use in any cryptographic system.

Security of the key agreement:

Mixing of bitwise Boolean operations (XOR), bitwise and

operations oriented (manipulation) serves to avoid purely algebraic attacks and the purely bit oriented attacks and prevents the mathematical behavior of the scheme from being shaped easily. They both contribute to a great mathematical complexity together with high computational efficiency. Moreover, only very efficient operations are used, bitwise Boolean operations, and bits displacements, and they are both of easy implementation either by hardware or by software.

This method generates pseudo random numbers by selecting a certain piece of secure information. For the users involved in this system to generate these random numbers, they must agree upon this information in advance to enable them to generate their own random numbers which are then used in cryptographic systems.

The users also agree upon the predefined structure of this secret information or the key. The structure in this method consists of a certain number of digits constituting certain digital data that can either be represented as characters or as a bit sequence. The agreement upon the size (n) of this key, which depends practically on the cryptographic system that is intended to be used with, represents one of the constraints of this structure. And another constraint is the method used for splitting this piece of information for the various operations. The secure information is dealt with at byte level segments. Obviously each byte contains different digital contents.

The algorithm generates the first pseudorandom key at the end of the first run, however to generate more random numbers, further runs can be done. Each of the successive runs takes the previously generated random key as the input. Therefore, the difficulty and complexity of the generated keys will increase. The acceptance strength of the generated random numbers will be decided by the tests that will be performed on the results in the following section.

V. Implementation and Results

A Computer program is written in C# language for the proposed PRNG algorithm illustrated in Fig. 1. It is designed to accept a seed of any number of characters and generate as many random numbers as practically required with any length of bit sequence. It has been experimented for the generation and testing of random keys of 64, 128, and 512 bits lengths. An example demonstrating the implementation steps for the algorithm execution and elementary randomness tests are included in the next subsection, then four elaborate tests were carried out and their results were listed in next subsection.

A. PRNG Implementation

The following example can illustrate the implementation of the scheme for a 64 bits key sequence, i.e. the seed consists of 8 characters. Let us assume that step 1 produce the binary representation of the seed characters to be:

"10001100 10101010 00110011 10010001 01000010 00100011 00010100 10100101".

Step 2 produces:

"00100110 10011001 10100010 11010011 01100001 00110111 10110001 00101001",

Then step 3 produces:

"10011001 00100110 11010011 10100010 00110111 01100001 00101001 10110001".

And finally step 4 will produce:

"00110111 01100001 00101001 10110001 10011001 00100110 11010011 10100010".

This is the generated pseudorandom number of the first round. Then more random numbers can be generated by repeating the PRNG algorithm using the generated number of a round as input to the algorithm for the next round.

To test the randomness of the generated random number, the following simple frequency test can be conducted. This test uses the chi-function formula given in (10):

$$\chi^2 = \frac{(n_0 - n_1)^2}{n} \tag{10}$$

where n_0 and $n_1$are the numbers of 0's and 1's in the generated key sequence, respectively. Good sequence in the generated random number should have χ^2 values in the range $0<\chi^2<3.84$[24].

Since χ^2 = (29-35)2/64 = 0.5625 which is < 3.84 for the above example, therefore it is considered as acceptable random number.

Counting the 0's and 1's in the generated key in the above example resulted into n_0 = 34 and n_1 = 30, therefore $\chi 2 = (34-30)^2/64 = 0.25$, and since this value is < 3.84, hence it is considered as acceptable random number.

Other test like frequency (monobit) can also be conducted, for the above example above.

The string length n = 64 and the sum of all number string bits sequence after converting 0's and 1's to -1 and +1, respectively, S_n is determined as

S_n =-1-1+1+1-1+……+1-1+1-1-1-1+1-1 = - 4.

Then from (1), the test statistic observed value

$S_{obs} = \left|-6\right| / \sqrt{64} = 0.5.$

Now applying (2), the P-*value* is calculated and it is here

P-value=erfc (0.5/8) = 0.9296

Then since 0.9296is > 0.01, the number is random otherwise it is not.

Another series test for the frequency of occurrence of sequences of two bits, i.e. 00, 01, 10, and 11 can be conducted. For the above example are n_{00}=16, n_{01}=16, n_{10}=17 and n_{11}=10 and the number of 1's and 0's in the key sequence n_1=30 and n_0=34.

Applying the chi-function for this test which is given by (11) [24]:

$$\chi^2 = \frac{4}{n-1}(n_{00}{}^2 + n_{01}{}^2 + n_{10}{}^2 + n_{11}{}^2) - \frac{2}{n}(n_1{}^2 + n_0{}^2) + 1 \tag{11}$$

The criterion for good randomness sequence is $\chi 2 \leq 5.99$. Applying (11) results into

$$\chi^2 = \frac{4}{64-1}(15^2 + 18^2 + 17^2 + 11^2) - \frac{2}{64}(30^2 + (-34)^2) + 1 = -10.96429 \leq 5.99$$

Therefore, this key passes the randomness test.

B. PRNG Testing

The potential problems with deterministic generators are their failure in statistical pattern-detection tests. These problems include lack of distribution uniformity, correlation of successive values; output sequence has poor dimensional distribution, and shorter seed state. The random sequences generated by the proposed PRNG are tested for randomness by the four tests processes outlined in Section III. Although, the tests were done for 64, 128, 1nd 512 bits key sequences, however, it can be practically done for any key length.

Table II lists the calculated percentages of successfully generated random key sequences when the used seed for generating random numbers is only numeric's, i.e. digits 0, 1, 2, … , 9. The tests included are explained four tests in section 3 namely; frequency (monobit) test, Frequency within a Block test, Runs test, and Longest-Run-of-Ones in a Block test. This table lists the results for three commonly required random key sequences; 64, 128, and 512 bits. However, Table III lists the percentages of successfully random keys generated using alphanumeric seed (i.e. both upper and lower case letters in addition to the numbers). It is also conducted for the same lengths and tests selected.

TABLE II
SUCCESSFUL RANDOMNESS FOR ONLY NUMERIC KEYS

Test	PERCENTAGE OF SUCCESSFUL RANDOMNESS (NUMERIC SEED ONLY) n=64n=128n=512		
Frequency (monobit)	73%	40%	13%
Frequency within a block	84%	80%	23%
The Runs test	91%	62%	23%
Longest run of 1's in a block	N/A	99%	99%

TABLE III
SUCCESSFUL RANDOMNESS FOR ALPHANUMERIC KEYS

Test	PERCENTAGE OF SUCCESSFUL RANDOMNESS (ALPHANUMERIC SEEDS KEYS) n=64n=128n=512		
Frequency (monobit)	96%	89%	82%
Frequency within a block	98%	99%	98%
The Runs test	100%	99%	89%
Longest run of 1's in a block	N/A	98%	98%

The results listed in this section have shown that the proposed PRNG algorithm effectively generates key sequences with a considerably acceptable randomness. The randomness tests showed that increasing the number of rounds to generate more keys enhances the efficiency of random key generation for cryptographic system applications. The scheme is characterized by its simple yet practical design as it does not have complicated and lengthy exponentiation processes. This leads to more efficient software and hardware implementations.

VI. CONCLUSION

An algorithm for computationally fast, cryptographically secure pseudorandom key generator has been proposed and described in this paper. It is based on mixing bitwise Boolean operations with bits manipulations and displacements for secret splitting. The implemented randomness tests have shown that the generated sequences were unpredictable and passed successfully stringent test suites. It obviously relies on the sender/receiver agreement protocol regarding the cryptosystem they are using, the key length, number of keys and way of their generation. Therefore the required random keys will be securely generated accordingly.

The algorithm was programmed in C# language on a 64 bits word length computer using only bitwise XOR and blocks exchanges. Hence excellent performance was achieved. Besides, the program produces as many number of random sequence keys as required, in which each random sequence is generated from a random one.

REFERENCES

[1] B. Schneier, "Applied cryptography: protocols, algorithms, and source code in C," Second Edition, John Wiley & Sons, 1996.
[2] D. Dilli, Madhu S., "Design of a New Cryptography Algorithm using Reseeding -Mixing Pseudo Random Number Generator," IJITEE, vol. 52, No. 5, 2013
[3] K. Marton, A. Suciu, C. Sacarea, and Octavian Cret, "Generation and Testing of Random Numbers for Cryptographic Applications," Proceedings of the Ramanian Academy, Series A, Vol. 13, No. 4, 2012, PP 368–377.
[4] S. Martain, "Testing of True Random Number Generator Used in Cryptography,"International Journal of Computer Applications IJCA, Vol.2, No. 4, 2012.
[5] Wikipedia, "Pseudorandom number generator", Last visited December 2014.
[6] D. Dilli, and S. Madhu, "Design of a New Cryptography Algorithm using Reseeding -Mixing Pseudo Random Number Generator," IJITEE, vol. 52, no. 5, 2013.
[7] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A Statistical Test Suite for Random and Pseudorandom Generators for Cryptographic Application," NIST Special Publication 800-22, 2001.
[8] P. Burns, "Lagged, Fibonacci Random Number Generators", GS 510, fall 2004, http://lamar.colostate.edu/~grad511/lfg.pdf.
[9] A. B. Orue, F. Montoya, and L. H. Encinas, "Trifork, a New Pseudorandom Number Generator Based on Lagged Fibonacci Maps," Journal of Computer Science and Engineering, vol. 1, no. 10, 2010.
[10] Random.org (Randomness and Integrity Service LTD),
[11] https://www.random.org/integers/, Last visited 3/1/2015.
[12] F. W. Burton, and R. L. Page, "Distributed random number generation", Journal of Functional Program, vol. 2, no. 2, 1992, PP 203–212.
[13] K. Claessen, and M. Palka, "Splittable Pseudorandom Number Generators using Cryptographic Hashing," Proceedings of Haskell Symposium, 2013, PP 47-58.
[14] J. M. Bahi, and C. Guyeux, "Topological chaos and chaotic iterations, application to hash functions," IEEE World Congress on Computational Intelligence WCCI', Barcelona, Spain, July 2010. Best paper award, PP 1–7,
[15] J. Bahi, C. Guyeux, and Q. Wang, "A novel pseudo-random generator based on discrete chaotic iterations," INTERNET'09, 1-st International conference on Evolving Internet, Cannes, France, August 2009, PP 71–76.
[16] J. Bahi, C. Guyeux, and Qianxue Wang, "A pseudo random numbers generator based on chaotic iterations; Application to watermarking," International conference on Web Information Systems and Mining, WISM 2010, vol. 6318 of LNCS, Sanya, China, October 2010, PP 202–211.
[17] Y. Hu, X. Liao, K. W. Wong, and Qing Zhou, "A true random number generator based on mouse movement and chaotic cryptography," Chaos, Solitons & Fractals, vol.40, no. 5, 2009, PP 2286–2293.
[18] L. De Micco, C. M. Gonzaez, H.A. Larrondo, M.T. Martin, A. Plastino, and O.A. Rosso, "Randomizing nonlinear maps via symbolic dynamics," Physica A: Statistical Mechanics and its Applications, vol. 387, no. 14, 2008, PP 3373–3383.
[19] L. Larger, and J. M. Dudley, "Nonlinear dynamics Optoelectronic chaos," Nature, vol. 465, no. 7294, 2010, PP 41–42.

[20] Q. Wang, J. Bahi, C. Guyeux, and X. Fang, "Randomness quality of CI chaotic generators; application to internet security," INTERNET'2010. The 2nd International Conference on Evolving Internet, Valencia, Spain, September 2010. IEEE Computer Society Press. Best Paper award, PP 125–130.
[21] H. B. Neumann, S. Scholze, and M. Voegeler, "Method of generating pseudo-random numbers," US 20090150467 A1 , Jun 11, 2009.
[22] M. N. Elsherbeny, and M. Raha, "Pseudo –Random Number Generator Using Deterministic Chaotic System," International Journal of Scientific and Technology Research," vol. 1, no. 9, Oct. 2012.
[23] S. Behnia, A. Akhavan, A. Akhshani, and A.Samsudin, "A novel dynamic model of pseudo random number generator," Journal of Computational and Applied Mathematics –J COMPUT APPL MATH, vol. 235, no. 12, 2011, PP 3455-3463.
[24] W. Bhaya and W. Mahdi, " Fingerprint Security Approach for Information Exchange on Networks," European Journal of Scientific Research, vol. 123, no 2, 2014, PP 169-181.

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.

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 Modeling.