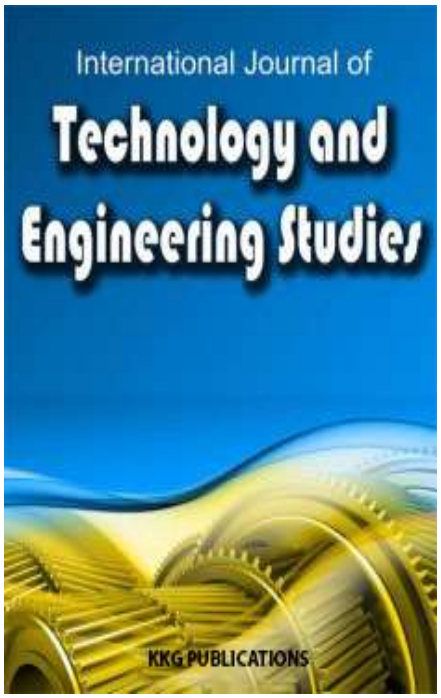


Key Knowledge Generation
Publication details, including instructions for author and
Subscription information:
http://kkgpublications.com/technology/

Using Deterministic Genetic Algorithm to Provide Secured Cryptographic Pseudorandom Number Generators

AMANIE HASN ALHUSSAIN

Peoples' Friendship University of Russia

Published online: 15 August 2015

To cite this article: A. H, Alhussain, "Using deterministic genetic algorithm to provide secured cryptographic pseudorandom number generators," *International Journal of Technology and Engineering Studies*, Vol. 1, no. 4, pp. 106-115, 2015.
DOI: https://dx.doi.org/10.20469/ijtes.40001-4

To link to this article: http://kkgpublications.com/wp-content/uploads/2015/12/IJTES-40001-4.pdf

KKG PUBLICATIONS

International Journal of Technology and Engineering Studies

IJTES

Vol, 1, no. 4, pp. 106-115, 2015

USING DETERMINISTIC GENETIC ALGORITHM TO PROVIDE SECURED CRYPTOGRAPHIC PSEUDORANDOM NUMBER GENERATORS

AMANIE HASN ALHUSSAIN

Peoples' Friendship University of Russia

Keywords:
Genetic Algorithm
Pseudorandom Number Generators
Deterministic Approach
Period
Graphical Test
Statistical Test
Frequency Test
Runs Test
Autocorrelation Test
Entropy.

Abstract. This research shows a method of providing Pseudorandom Number Generator (PRNG) without the properties of periodicity and predictability, i.e., secured cryptographic PRNG, by using a deterministic genetic algorithm. PRNGs are so important in cryptography. Their main advantages of PRNGs are speed, efficiency, and reproducibility. This article studies the properties of uniformity, randomness, and independence between two sequences of random numbers. The first sequence is generated by using a traditional pseudorandom number generator (PRNG). In contrast, the second one is generated with the help of a cryptographic pseudorandom number generator, which is modified by a genetic algorithm (GA). This work shows the graphical and statistical tests: frequency test, runs test, Autocorrelation test, and entropy. The tests are performed and implemented with the help of three programs: MATLAB, Minitab, and IBM SPSS Statistics. This statistical study has shown that the proposed deterministic GA improves the random numbers generated by conventional PRNG, i.e., it provides secured cryptographic pseudorandom number generators.

Received: 04 August 2015
Accepted: 02 October 2015
Published: 05 December 2015

INTRODUCTION

The problem of generating random numbers on electronic computers existed a long time ago, but with the development of cryptography, it received an additional interest. The need of deterministic random numbers in cryptography is increased; they can be applied in establishing session keys, private keys, and asymmetric schemes, when signing documents, secret sharing schemes and key generation.

There are two types of random number generators – true and pseudo. The fundamental difference between the two types is that the true random number generators sample is a source of entropy whereas pseudorandom number generators (PRNGs) instead use a deterministic algorithm to generate numbers [1].Each of them has its advantages and disadvantages. Generally the limitation of one type is the advantage of the other.

True random number generators do not exhibit periodicity, and in them there is no dependence between the generated numbers; however, their main disadvantages are being expensive, slow, inefficient, and that a sequence of numbers cannot be reproduced. Pseudo-random number generator (PRNG) is fast, efficient and reproducible; however, their main disadvantages are the periodicity and predictability of random numbers based on knowledge of previous sequences; this leads to a low level of security when used in cryptography [2].

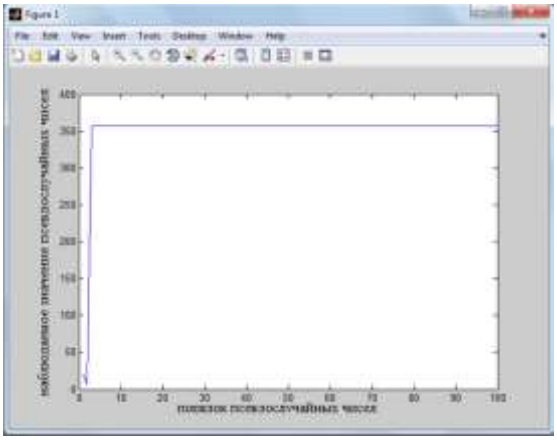


Fig. 1. The length of the period, which is generated by the linear congruential generator, is equal to 1

*Corresponding author: Amanie Hasn Alhussain

E-mail: amanie-alhussain@hotmail.com

Any pseudorandom number generator (PRNG) would be always sooner or later "loop", i.e. the numbers form a loop which is repeated an infinite number of times. Repeating the cycle is called the period [2,3]. The length of the period in the sequence has different values. For example, it can be just one value, as shown in fig.1 and fig.2; or it may be two values, as shown in fig.3 and fig.4 etc...

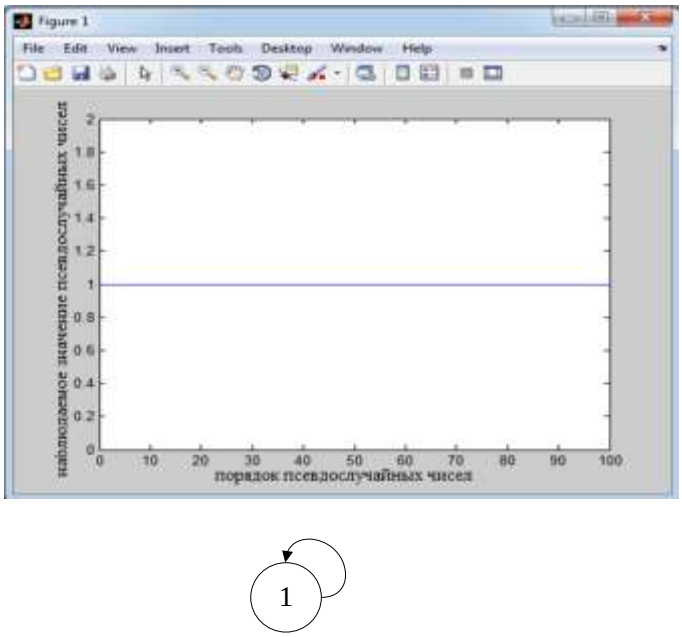


Fig. 2. The length of the period, which is generated by Blum-Blum-Shub Generator, is equal to 1.

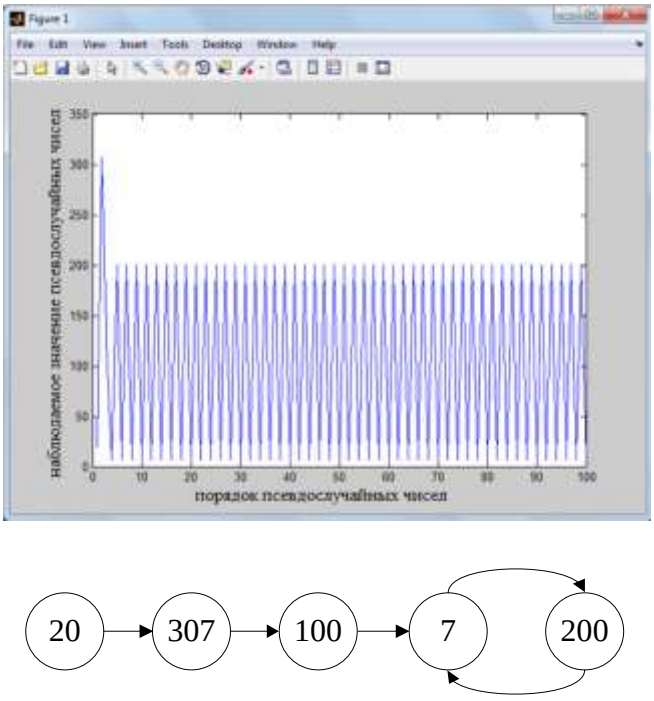


Fig. 3. The length of the period, which is generated by Quadratic Congruential generator, is equal to 2

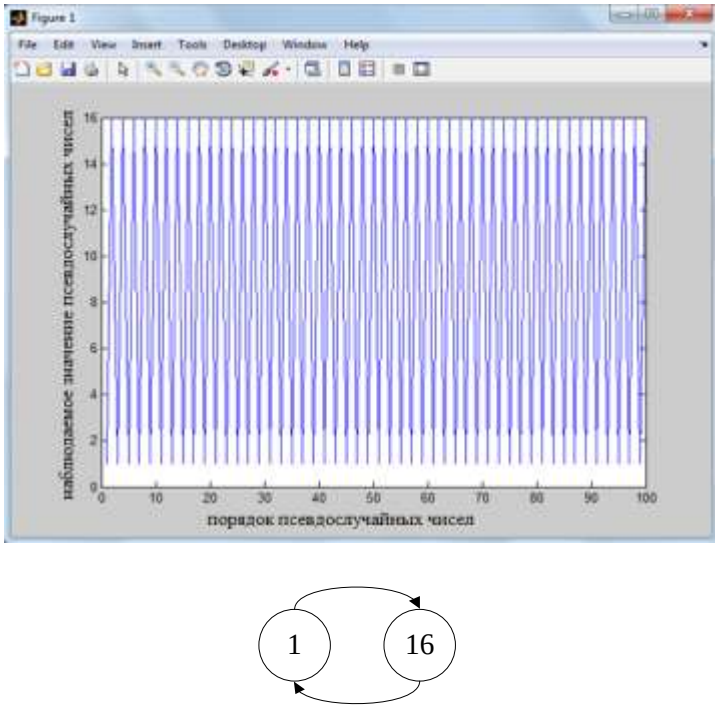


Fig. 4. The length of the period, which is generated by Quadratic Congruential generator, is equal to 2

Many ciphers like the Vernam cipher, XOR [4], Vigenère and the others require a lot of keys [6]. A significant disadvantage of these ciphers is where to store a huge amount of keys and how to move them [7,8]. To overcome these shortcomings, they use pseudorandom number generators. Fig. 5 shows the schema of XOR encryption algorithm. This schema shows clearly the main role that PRNG plays.

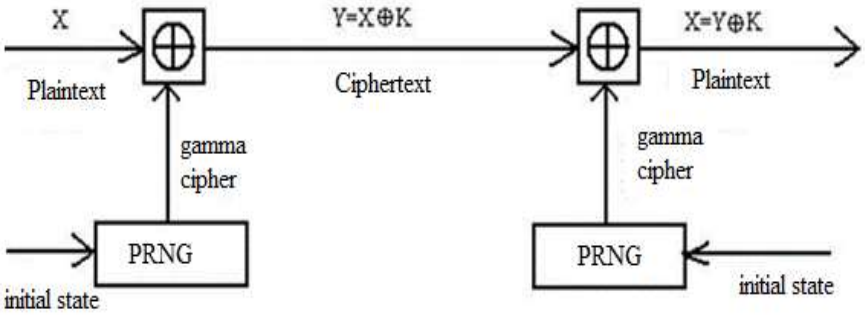


Fig. 5. Schema of XOR encryption

The main purpose of this paper is showing how to increase the security level of pseudorandom number generators by deleting the properties of periodicity and predictability which leads to increasing the security level of ciphers that depend on these PRNG, like XOR encryption [4].

The rest of the paper is organized as follows: 1st section displays the proposed genetic algorithm that is used to improve the randomness properties of PRNG, the 2nd presents the graphical test of the proposed PRNG, the 3rd section shows the statistical tests of the proposed method, the conclusion concludes the paper [3].

THE PROPOSED GENETIC ALGORITHM

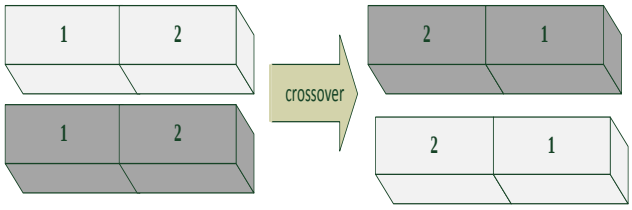
The proposed genetic algorithm has the following features and operators:

1) The representation of chromosome in the proposed method is binary (0 or 1).
2) The length of the chromosome should be defined by the user, (i.e. dynamic length, this gives an extra level of security for the generators).
3) The size of the population should be defined by the user, (i.e. dynamic length, this gives an extra level of security for the generators).

4) Genetic operations: crossover and mutation.

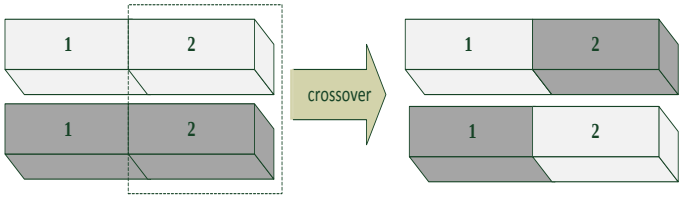
- ✓ Crossover: the type of crossover is one-point crossover according to the selected size of the chromosomes.
- ✓ Mutation: invert deterministically some of selected "genes" to avoid repeating numbers. (Convert 0 into 1 or vice versa).

The chosen crossover in the proposed method included five types, as follows:



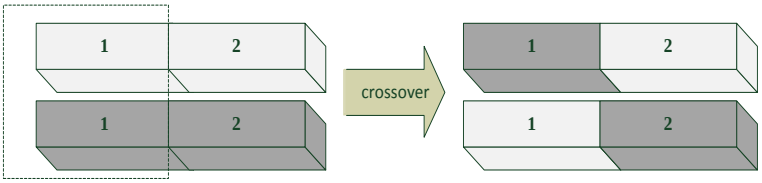
Crossover Vertical Right

Cross vertically the second part of the first chromosome with the second part of the second chromosome.



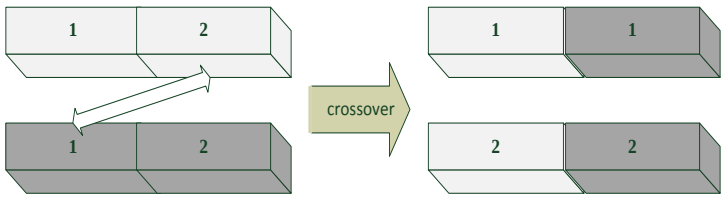
Crossover Vertical Left

Cross vertically the first part of the first chromosome with the first part of the second chromosome.



Crossover Diagonal Right

Cross diagonally the second part of the first chromosome with the first part of the second chromosome.

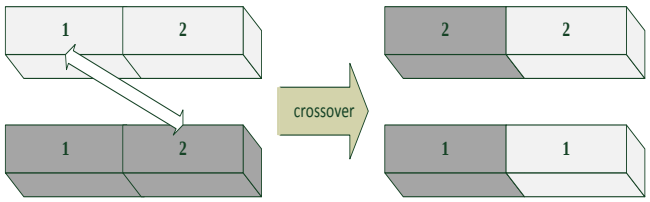


Crossover Diagonal Left

Cross diagonally the first part of the first chromosome with the second part of the second chromosome.

Crossover Diagonal Left and Right

Cross diagonally the first part of the first chromosome with the second part of the second chromosome and the second part of the first chromosome with the first part of the second chromosome.



If the crossover type would be applied sequentially, the generated sequence of PRNG would include period, i.e. the applying of the crossover, it should not be in order;

The algorithm must be written and designed so that it cannot be predictable (containing period) but reproducible. Therefore, the proposed genetic algorithm is designed to use all types of crossover in disorder. This is done by using the following equation:

Type of crossover = (the order of chromosome$_1$ in population + the order of chromosome$_2$ in population + the order of population which contains crossed chromosomes) mod 5.

If the type of crossover = 1, apply crossover diagonal left and right.

If the type of crossover = 2, apply crossover diagonal left.

If the type of crossover = 3, apply crossover diagonal right.

If the type of crossover = 4, apply crossover vertical right.

If the type of crossover = 0, apply crossover vertical right.

Fig.6 shows the schema of the proposed genetic algorithm.

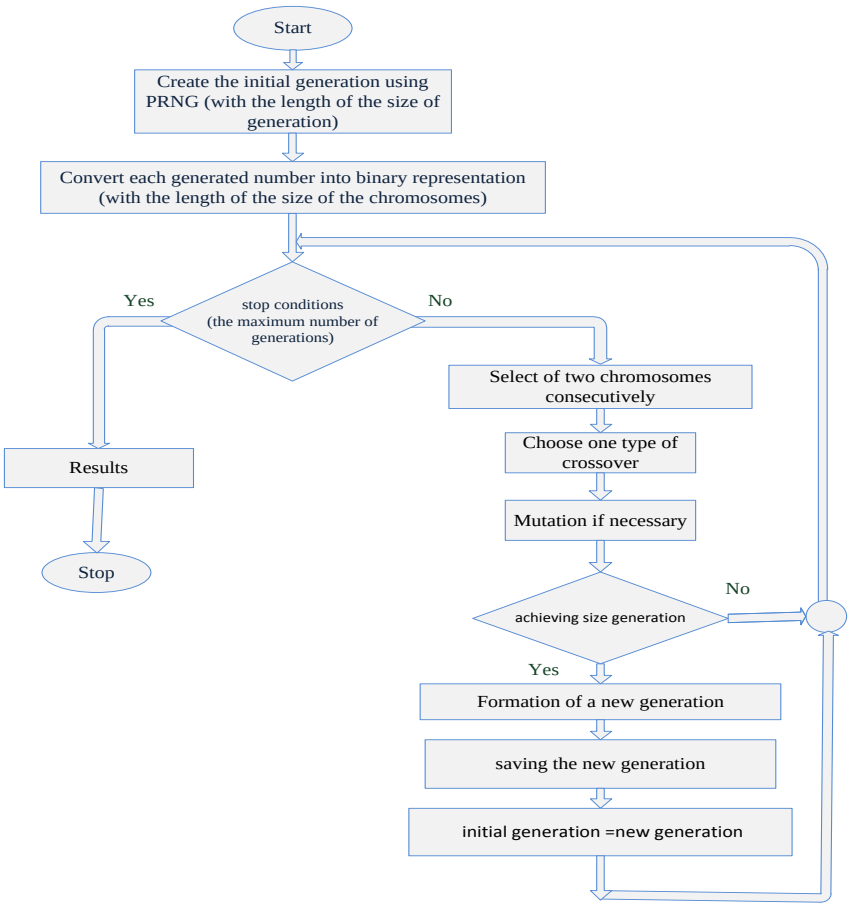


Fig. 6. The scheme of the proposed genetic algorithm

GRAPHICAL TEST

The following tests are applied by using Linear Congruential generator with the parameters: Multiplier=7, increment=10, modulus=988, initial value=7, generation size=100. Graphical comparison between the two sequences of Linear Congruential Generator before and after using genetic algorithm, and plotted by MATLAB, is shown in fig.7 and fig.8:

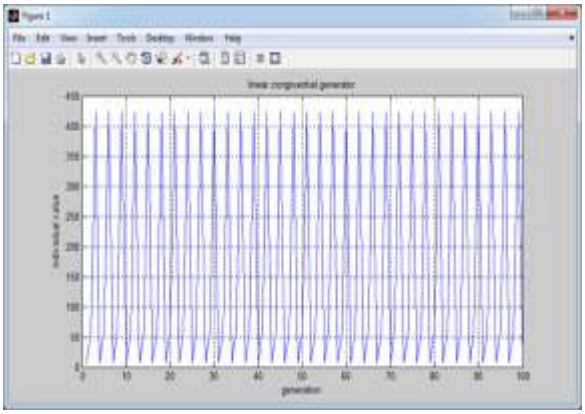


Fig. 7. The sequence generated by linear congruential generator without using GA

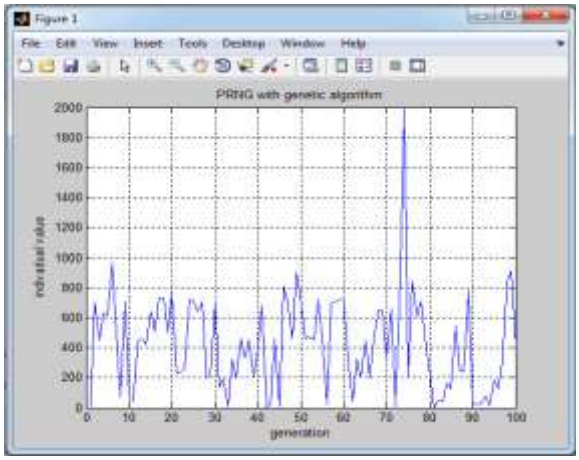


Fig. 8. The sequence generated by Linear Congruential Generator using GA

Histogram Comparison

Histogram comparison between two sequences of Linear Congruential Generator before and after using genetic algorithm, and plotted by Minitab program, is shown in fig.9 and fig.10:

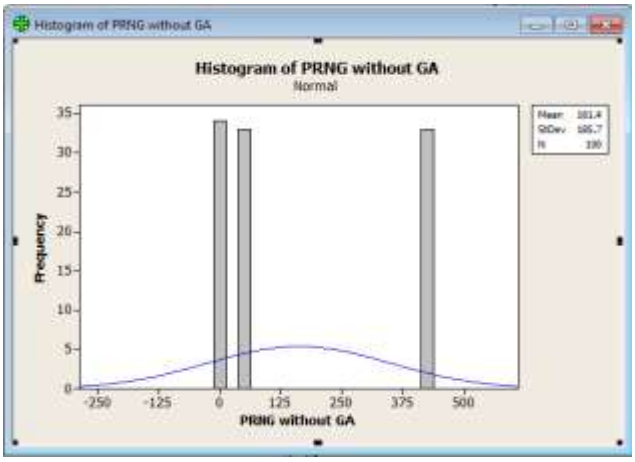


Fig. 9. The Histogram of Linear Congruential Generator without Using GA

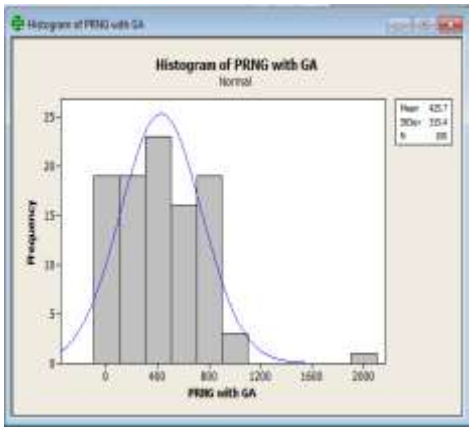


Fig. 10. The histogram of Linear Congruential Generator using GA

STATISTICS TEST

This paper highlights three properties uniformity, independence and randomness. The first test uses uniformity, the second and third ones test independence, while the fourth tests the randomness.

1. Frequency test
2. Runs test
3. Autocorrelation test
4. Entropy.

Frequency Test

The frequency test is a test of uniformity. It is applied by using Kolmogorov-Smirnov test which measures the agreement between the distribution of a sample of generated random numbers and the theoretical uniform distribution.

Null and Alternative Hypothesis

H_0: data follow a normal distribution, H_1: data do not follow a normal distribution, Significance level: $\alpha = 0.01$.

Applying Kolmogorov-Smirnov test on the sequence of numbers which is generated by Linear Congruential Generator without using GA, and tested by Minitab program is shown in fig.11:

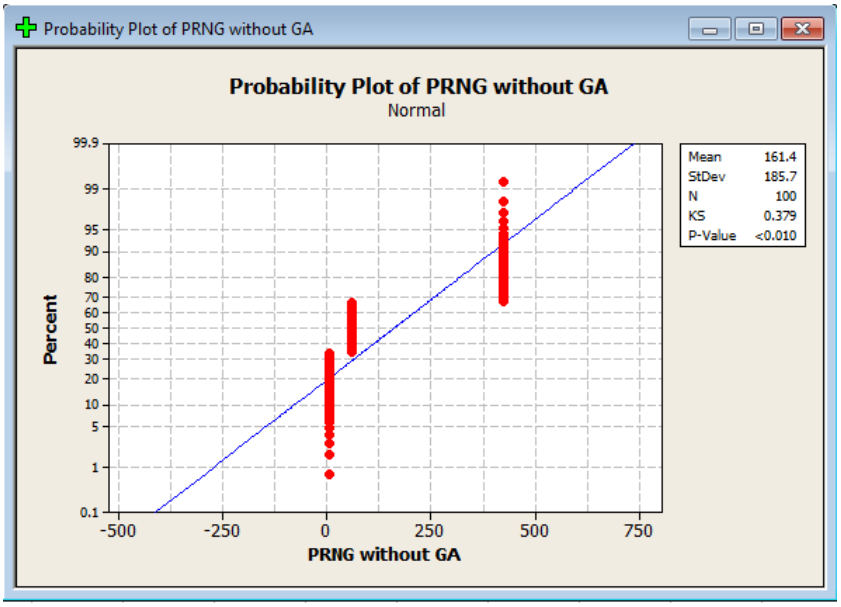


Fig. 11. Kolmogorov-Smirnov test of Linear Congruential Generator without Using GA

The test concludes: $p < \alpha$=0.010 then we reject the null hypothesis that the data follow a normal distribution, and accept the alternative hypothesis H_1 that data do not follow a normal distribution.

(P-value (the probability value) is the value p of the statistics used to test the null hypothesis. If $p < \alpha$ then we reject the null hypothesis).

Applying Kolmogorov-Smirnov test on the sequence of numbers generated by PRNG using GA, and tested by Minitab program is shown in fig 12:

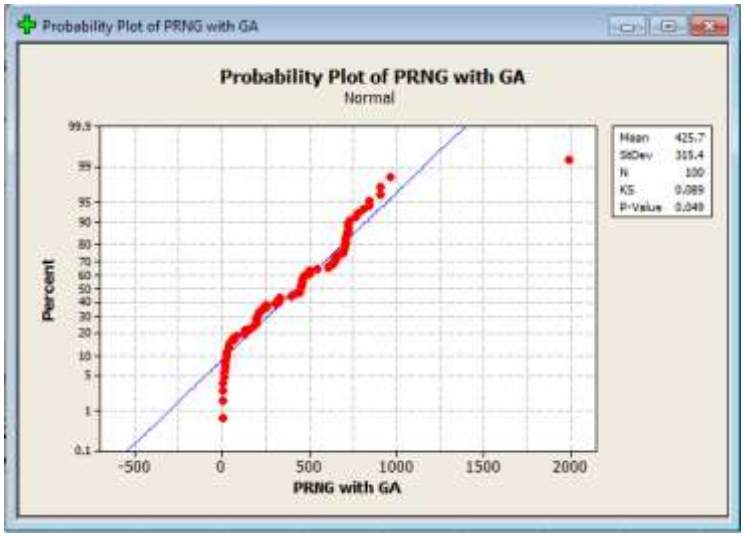


Fig. 12. Kolmogorov-smirnov test of Linear Congruential Generator using GA

The test concludes: (P=0.049) > (α=0.010) then we accept the null hypothesis that data follow a normal distribution.

Runs Test

Runs test tests the number of runs above and below some constant (usually the mean). The test involves counting the actual number of occurrences of runs of different lengths and comparing these counts with the expected values using a Chi-square. The runs test examines the arrangement of numbers in a sequence to test the hypothesis of independence.

Null and Alternative Hypothesis

H_0: the sequence was produced in a random manner, H1: the sequence was not produced in a random manner**.** Significance level: $\alpha = 0.01$**.**

Applying runs test on the sequence of numbers which is generated by Linear Congruential Generator without using GA, and tested by Minitab program is shown in fig 13:

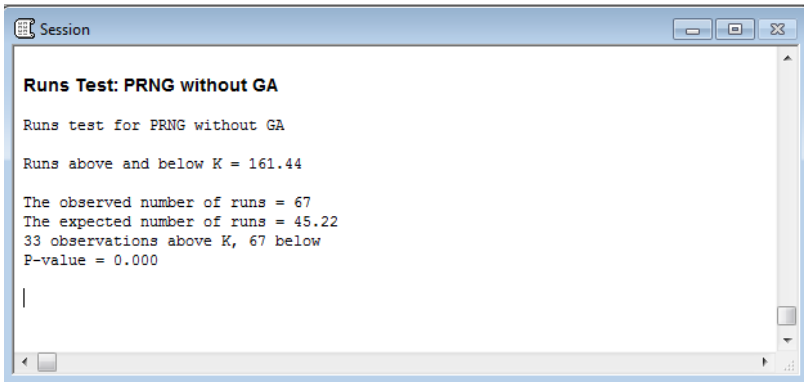


Fig. 13. Runs test of Linear Congruential Generator without using GA

The test concludes: (p =0.000) < (α=0.010) then we reject the null hypothesis that the sequence was produced in a random manner, and accept the alternative hypothesis H_1 that the sequence was not produced in a random manner.

Applying runs test on the sequence of numbers generated by PRNG using GA, and tested by Minitab program is shown in fig 14:

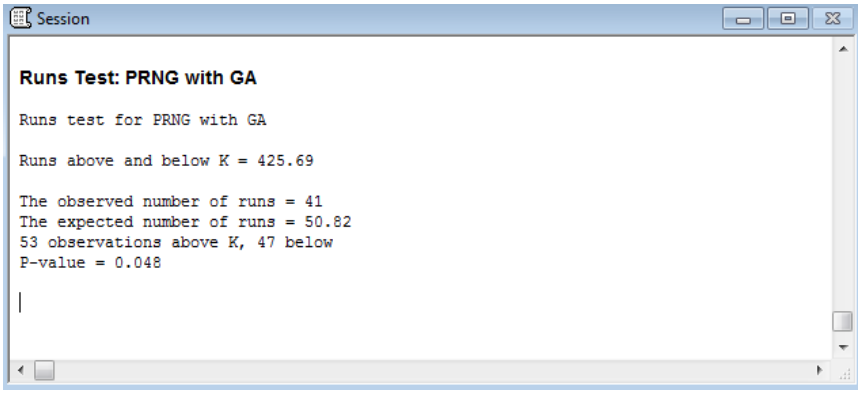


Fig. 14. Runs test of Linear Congruential Generator using GA

The test concludes: (p =0.048)> (α=0.010) then we accept the null hypothesis that the sequence was produced in a random manner.

Autocorrelation Test

The tests for auto-correlation are concerned with the dependence between numbers in a sequence. It can be used to detect non-randomness in data. It is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal.

Null and Alternative Hypothesis

H_0: $\rho = 0$ (The null hypothesis states there is no relationship between the two sequences of numbers).H_1: $\rho \neq 0$(alternative hypothesis states that there is relationship between the two sequences of numbers): ρ denotes the correlation coefficient of the population.

Significance level: $\alpha = 0.01$. Applying autocorrelation test by Minitab program on the variables of Linear Congruential Generator without GA, and with GA is shown in fig.15:

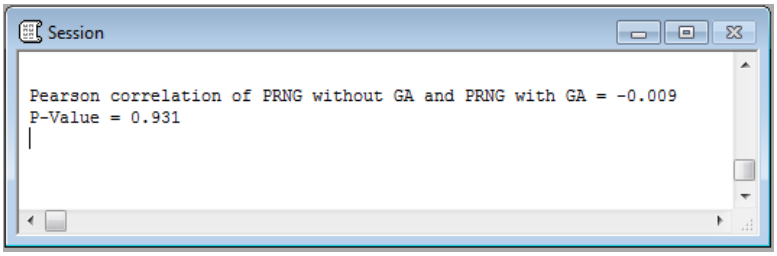


Fig.15. Auto-correlation test of two sequences (Linear Congruential Generator with and without GA)

The test concludes: (P-value=0931) > (α=0.010) then we accept the null hypothesis ($\rho = 0$):i.e. there is no relationship between the two sequences of numbers. r=-0.009 which indicates that there is no relationship between the two variables or the generated sequences of numbers by returning to the table illustrated in [5].

Entropy Test

The entropy as a measurement has the concept of disorder or unpredictability of the information elements. The higher it is, the more chaotic, unpredictable and redistributed the information is.

Comparison of the results of applying the entropy as a measure of randomness, demonstrated by MATLAB package on the sequence of a Linear Congruential generator without the use of a genetic algorithm is shown in fig.16 and using a genetic algorithm is shown in fig. 17.

```
>> Entropy_Linear_Generator_Without_GA=Entropy(linearGeneratorWithoutGA)

Entropy_Linear_Generator_Without_GA =

    1.5848
```

Fig. 16. The entropy as a measure of randomness of linear congruential generator without using GA

```
>> Entropy_Linear_Generator_With_GA=Entropy(linearGeneratorWithGA)

Entropy_Linear_Generator_With_GA =

    6.3688
```

Fig. 17. The entropy as a measure of randomness of linear congruential generator using GA

From the entropy test in Figure 16 and 17 it is concluded that (the entropy of a linear congruential generator without the use of GA = 1.58) < (entropy of a linear congruential generator with GA = 6.37). I.e. the sequence of linear congruential generator with GA is more chaotic and unpredictable than the sequence without using GA.

CONCLUSION AND RECOMMENDATIONS

This paper demonstrates the ability to generate secured cryptographic pseudorandom number generators using a genetic algorithm, which is aperiodic and reproducible.

This statistical study has shown that the proposed deterministic genetic algorithm, improves the random numbers which are generated by conventional pseudorandom number generator (PRNG), i.e. it provides secured cryptographic pseudorandom number generators.

This sequence, which is generated using secured cryptographic pseudorandom number generators, satisfies the following important properties:

- provides uniformity: a sequence of normally distributed (in accordance with the test of frequency)
- Ensures independence: confirmed for a sequence of randomly distributed (in accordance with the runs test) and shows no correlation between the elements of the sequence (according to autocorrelation test).
- Improves the randomness (in according with the entropy test).

Declaration of Conflicting Interests

There are neither financial nor non-financial conflicts involved in carrying out this work.

REFERENCES

[1] C. Kenny and K. Mosurski, *Distributed Systems Group-Random Number Generators: An Evaluation and Comparison of Random. org and Some Commonly Used Generators.* Trinity College Dublin, 2005.

[2] H. Al-Hussain and V. L. Stefanuk, "Improvement of randomness level of pseudorandom number generators in cryptography," in *2nd International Scientific Conference on Theoretical and Applied Sciences*, pp. 172-177. US, Cibunet Publishing, New York, USA, 2015.

[3] H. Al-Hussain and V. L. Stefanuk, "Using Genetic Algorithm to improve periodic level of pseudorandom number generators," in *1st European Conference on Informational Technology and Computer Science*: *East West*, pp. 25-34, Vienna, Austria, 2015.

[4] H. Al-Hussain and V. L. Stefanuk, "Using deterministic genetic algorithm to increase the security level of XOR encryption," «Приоритеты мировой науки: эксперимент и научная дискуссия»: Материалы VIII международной научной конференции: CreateSpace ,с. 15-18,17-18 июня г. – Южная Каролина, Северный Чарльстон, США, 2015.

[5] "Statistics How to Statistics for the Rest of Us." http://www.statisticshowto.com/how-to-compute-pearsons-correlation-coefficients/

[6] I. Cicek, A. E. Pusane, and Dundar, G. "A novel design method for discrete time chaos based true random number generators, "*Integration*, vol. 47, no. 1, pp. 38-47, 2014.

[7] G. Chen, "Are electroencephalogram (EEG) signals pseudo-random number generators?" *Journal of Computational and Applied Mathematics*, vol. 268, pp.1-4, 2014.

[8] A. Beirami, and H. Nejati, "A framework for investigating the performance of chaotic-map truly random number generators." *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 7, pp.446-450, 2013.

[9] O. Reyad, and Z. Kotulski, "On pseudo-random number generators using elliptic curves and chaotic systems." *Applied Mathematics & Information Sciences*, vol. 9, no. 1, p.31-35, 2015.

— This article does not have any appendix. —

KKG PUBLICATIONS