Analysis and Comparison of Pseudo Random Number Generator Algorithms

Luke Fetchko
Department of Computer Science
University of South Carolina Upstate
Spartanburg, South Carolina, USA
lfetchko@email.uscupstate.edu

ABSTRACT

As computer programming and engineering has rapidly advanced over the previous decades, a new necessity for many applications has been discovered. This discovery is the need for a way to generate randomness in computer programs, different games with random elements, and in computer simulations. Another field of study interested in generating secure and unpredictable random numbers is cryptography. Random number generation for cryptography differs from other applications since these numbers must strictly adhere to security standards and be almost completely unpredictable. Computer programs and hardware must be explicitly told what to do based on algorithms. If an algorithm is known, then the results of this algorithm with predefined parameters will be predictable and this is not favorable for cryptography applications. Computer scientists needed to develop a method of random number generation that is efficient and reliable. With the advent of Pseudo Random Number Generator programs and algorithms, this began the journey and long story on the way to solving this important problem. In this research, it will focus on two primitive Pseudo Random Number Generation algorithms known as the Linear Congruential Method and the Middle Square Method. This research involved the implementation of both mentioned PRNG algorithms and conducting experimental tests on these algorithms to determine which of the algorithms is a better choice based on the evaluation metrics of execution time and the p-value from the Runs Test for Randomness. After analyzing the results of the experiment, the Linear Congruential Method performed better in execution time and passed all tests for the Runs Test for Randomness. Therefore, the Linear Congruential Method is the better technique to efficiently generate random numbers.

Keywords

PRNG – Pseudo Random Number Generator

LCM – Linear Congruential Method

MSM – Middle Square Method

TRNG – True Random Number Generator

IDE – Integrated Development Environment

1. INTRODUCTION

Randomness is a property needed for many different applications in the fields of science, mathematics, and technology. Computer programs or simulations that require randomness depend on certain means of producing numbers that are either random or appear to be random. Randomness usually entails unpredictability if truly random results are desired. The advent of random number generators attempted to provide a method of producing sequences of random numbers where necessary. There are two distinct methods of producing random numbers. These methods are the use of True Random Number Generators and the use of Pseudo Random Number Generators [2].

True Random Number Generators are different from PRNGs in the sense that the numbers produced by TRNGs are truly random and are not deterministic. Results from TRNGs should not be able to be replicated under any circumstance. TRNGs require a source of entropy that is completely unpredictable in order to generate truly random sequences. Randomness of a source is measured by the amount of entropy or unpredictability that specific source has. Examples of sources of randomness include radioactive decay, atmospheric noise, and movements of a computer mouse [2]. Measuring and analyzing these sources of randomness often requires highly advanced technologies to quantify the results. Efficiency is an issue while using TRNGs to generate random numbers as these measurements needed take large amounts of time to collect and analyze. As a result, TRNGs are expensive and not practical to implement in certain situations. There are applications that must use TRNGs as it is vital to have results that cannot be reproduced. These applications are cryptography and gambling [1]. If the method of generation is predictable, then individuals can crack the algorithms and reliably predict the outcomes. This would be detrimental to computers and computer systems in the field of cryptography [1].

Pseudo Random Number Generators are distinct from TRNGs since they are not truly random. PRNGs are classified as deterministic. PRNGs start with an initial seed value which is used as input for an arithmetical process (or algorithm) of generating random numbers. Classification as deterministic is the result of the use of this initial seed value. If a PRNG with a predetermined algorithm to produce results is used, then if we run multiple tests with the same initial seed value, we will encounter identical results. In some applications, this feature is useful and desired if a certain sequence of random numbers is needed at a later time in a particular application. Modeling and simulation applications benefit from this property of determinism. Another important property of PRNGs is efficiency. PRNGs are typically much faster at producing results compared to TRNGs. Applications requiring fast generation of random numbers greatly benefit from the efficiency of PRNGs.

The main goal of this study is to determine which algorithm for generating pseudo random numbers best approximates the properties of truly random numbers. Using some of the earliest studied and known PRNG algorithms this study will entail implementing these algorithms and running experimental tests to determine results and make a conclusion of which algorithm best approximates true random numbers. The algorithms that will be implemented and tested are the Middle Square Method and the Linear Congruential Method. Using the evaluation metrics of execution time and the p-value from the Runs Test for Randomness, an analysis of results will be conducted, and a decision will be made deciding on the most effective PRNG algorithm.

This study is relevant in today's world as the use of modeling and simulation techniques to solve problems spans across many disciplines. Monte Carlo simulations or methods require random number generation and reliable PRNGs are important to these applications [1]. Simulation models are used in computing, biomechanics, business, and economics to name a few. By researching and implementing different PRNG algorithms, it will provide a valuable insight into how PRNGs are created, tested, and evaluated. These skills and insights will be applicable to applications needing deterministic and efficiently generated random number sequences.

2. LITERATURE REVIEW

This study's primary focus is to implement and compare three different PRNG algorithms and based on evaluation metrics decide which algorithm is the most effective at approximating truly random numbers. Many studies have been conducted on PRNGs in the past that typically focus on one approach or technique and conducting intense statistical and mathematical analysis on the results of one certain approach. Computer scientists and engineers have worked on creating and improving upon PRNG algorithms for many years. Their goal is to continuously improve the results of generating numbers which appear to be truly random. Many studies conducted that focus on one approach and do heavy statistical analysis on the results leave the researchers and observers with many questions at the end. These studies may conclude that statistical results are not desirable or that generating truly random numbers may be possible with the current approach [3].

A previous study conducted on random number generation using bioelectrical and physical signals followed this pattern. This study provides great background information on random number generators including both PRNGs and TRNGs. This article clearly lays out advantages and disadvantages of both approaches to generating random numbers. As I mentioned, this study explains that TRNGs are commonly much slower than PRNGs and are largely hardware dependent. This study goes into great detail on the general structures of PRNGs and TRNGs. Diagrams are provided for easily visualizing the components of random number generators. Explanation of the bioelectrical and physical signals used in the random number generation technique is elaborate and valuable to understanding the overall operation of the random number generator. Data used for this study included EEG, EMG, EOG, blood volume pulse, galvanic skin response, and respiration readings [3]. This raw data was then transformed into binary number representations using a method involving modulo arithmetic [3]. Many random number generators appear random due to modulo arithmetic. Results of this study were analyzed using the NIST testing suite, scale index, and autocorrelation tests. In this study I will also conduct autocorrelation tests on the results of my experiments. The conclusion of the mentioned study on bioelectrical and physical signals concluded that the numbers obtained are suitable for use in many areas including game programming, authentication, and simulation [3]. The mentioned study also concluded that it is possible to generate truly random numbers using this approach. The mentioned study provided valuable information on many aspects of random number generation and was unique in the sense that bioelectrical and physical signals were used as sources of randomness then transformed and applied into a random number generator that is valuable for many applications.

When reviewing past research on the topic of random number generation, mentioning chaotic maps and chaos theory was common in these articles. Chaotic maps can potentially simultaneously produce and use entropy [1]. As chaotic maps are measurable dynamic systems, they can be studied analytically [1]. Chaos theory is interested in predicting outcomes of behaviors of inherently unpredictable systems. An example of a dynamic chaotic system is the weather system of our planet. Many random number generators make use of chaotic maps in their implementations. Chaos theory and chaotic maps are outside of the scope of this study but are worth mentioning and briefly describing as many articles on random number generators make use of these principles.

Due to time constraints and available resources, the research conducted for this study is limited, but valuable. Peer reviewed journal articles and conference papers relating to PRNGs are used for reference and the advancement of knowledge in this study. Locating scholarly or peer reviewed journal articles and conference papers on this topic proved to be difficult as many articles of interest require payment to view. With that said, valuable articles and studies open for public viewing have been found and utilized in this study. The current approach of researching the Internet and database collections is advantageous in terms of finding articles that are valuable to this study and leaving out research that may not be valuable. Disadvantages of the current approach include limited availability of articles of interest, and not making use of physical databases such as libraries and books.

3. METHODOLOGY

This study is focused on the comparison and analysis of multiple Pseudo Random Number Generator algorithms. These algorithms are the Linear Congruential Method and the Middle Square Method for the generation of pseudo random numbers. These are pseudo random numbers being generated from these algorithms since they appear to be random, but they are not truly random as mentioned earlier in the discussion of True Random Number Generators [2]. In order to accomplish the true random number generation, there must be a reliable source of entropy that can be used. The advantage of pseudo random number generation is that the numbers are generated much faster than from True Random Number Generators, and when starting with the same seed value the results can be replicated since that is useful in certain applications [1]. Also, the expenses associated with the TRNGs are often high as they involve expensive hardware and advanced engineering knowledge.

The first algorithm to be examined in more detail is the Middle Square Method. The Middle Square Method is one of the earliest known techniques for pseudo random number generation. There is not an equation for computation to describe the algorithm like many other pseudo random number generation techniques.

The algorithm itself is: pick a starting seed value that contains N digits, then square the seed value to obtain a result which contains N*2 digits, and then select the middle N digits from the square result and use that selection as the seed for the next iteration. Selection of the middle N digits at the end of the current iteration gives the resulting random number. This method of random number generation appears simple and that it would be a sufficient way to generate random numbers, but there are a few issues with this technique. One major problem with this technique is the possibility of repeating numbers, such as if the selected digits are 5 and 0. If we square the number 50, the resulting number is 2500 which has the same middle digits 5 and 0 as the previous iteration. This would result in the random numbers having repeated values until the maximum number generation value is reached. With this situation, it would not be a very useful method to compute random numbers as we would have an unwanted pattern of the same result as the random number for many iterations. In order to avoid this situation, to use the algorithm as desired would greatly depend on the choice of the seed value. This brings up another major issue that the algorithm heavily depends on the seed value entered. Ideally pseudo random number generator algorithms should not depend on the seed value or have the results altered in such a dramatic way based on seed value choice.

Next this study will examine the other pseudo random number generation method previously mentioned. This method is known as the Linear Congruential Method. Unlike the Middle Square Method, the Linear Congruential Method utilizes an equation in order to compute the random number values. The equation is as follows:

$$X_{i+1} = aX_i + c \ \ mod \ \ m$$

Here in the equation above there are multiple variables that are important to recognize and choose values that will return desired results. First, the variable X represents the iteration of the random number values. It can be noted that this equation uses the previous iteration's result to calculate the current iteration's resulting random number. Variable *a* in this equation is the multiplier value. Variable *c* in the equation is the increment value and variable *m* is the modulus value. The way this equation works is by first multiplying the previous iteration's random number value by the multiplier value. Then add the increment value to the result of the product, and finally do modulus with the sum result and the modulus value to obtain the random number result for the current iteration. Choosing good values for the variables is also important in this method. If one chooses certain values, then the resulting random numbers can form a pattern which is undesirable. An obvious pattern in the results of a random number generator raises a red flag since this pattern can be recognized which gives away hints to ultimately cracking the random number generation algorithm. It is ideal to choose the seed value, modulus value, increment value, and multiplier value to get a period approximately equal to the modulus value. Values of random numbers that exhibit the repeating pattern are not useful where random numbers are needed in various applications. The values that will be used for the implementation of this method will be 4294967296 (2^32) for the modulus value, 16843009 for the multiplier value, and 826366247 for the increment value. These parameters used come from the cc65 cross development package and are ideal for the purposes of this study.

Advantages of the Middle Square Method are that it is simple to implement and easy to understand, but it has multiple shortcomings for use as a pseudo random number generation method. As previously mentioned, the results of the Middle Square Method generator depend far too much on the initial seed value provided for the generator. Additionally, there is a chance to run into a repeating cycle of generating the same random number continuously. Another identifiable problem with the Middle Square Method is that once the middle N digits to be selected are all 0, then the rest of the random number resulting values will also be 0. Advantages of the Linear Congruential Method are that when the appropriate parameter values are chosen, there is a much longer period that is well known. When the period is longer in a pseudo random number generation technique, the resulting random numbers appear to be more random than with a shorter period. A disadvantage of the Linear Congruential Method is that if parameter values are chosen in such a way that results in a repeating pattern in the resulting random numbers, then the random numbers will begin to appear less random and once again may give away hints or clues to the underlying algorithm used to generate the random numbers. This would eradicate the principle of generating random numbers and could give away information or logic to users that was not intended to be given away by the developers.

4. IMPLEMENTATION

For the implementation of both pseudo random number generator algorithms, the programming language being used is Python. To be more specific, the version of Python being used is Python 3.9. One Python script will be used which will contain functions for each of the algorithms to be analyzed. Code of the algorithms will be executed in the IDLE environment which comes bundled with Python upon installation. This is the Integrated Development and Learning Environment which is the standard IDE for Python programmers.

In the implementation, a Python module named timeit is used and the timeit() function is called to analyze the execution time of both methods. This function accepts the PRNG function to be tested and the needed arguments for the PRNG function and times how long it takes to execute the algorithms a specified number of times. The time_function() averages these times and returns the average in milliseconds. Additionally the statsmodels.sandbox.stats module is used to conduct the Runs Test for Randomness and returns the z-score and the corresponding p-value.

Two more Python modules are used in the implementation. These modules are NumPy and matplotlib. NumPy is used to create an evenly spaced range for setting up the graph that is generated to analyze the execution times and make a comparison. Matplotlib is utilized to set up a bar graph which compares the execution times of each algorithm for each different user defined seed value.

5. EXPERIMENTAL SETUP

To set up the experiment for this study, the Python script will be loaded into the IDLE. By default, the script is programmed to generate 50 random numbers. For the experiment, the default value of 50 random numbers to be generated will be used. Each algorithm will be tested a total of 5 times. For both algorithms, these tests will use user defined seed values which will be noted. Each run of either algorithm will generate the default 50 random numbers. After running the 5 tests on either algorithm, the resulting random numbers list for the specific seed is passed to the

Runs Test for Randomness function. To further explain for clarity, the results of the 5 tests on the LCM algorithm using user defined seed values will be analyzed by passing the resulting list for each seed to the Runs Test function, and finally the results list of the 5 tests on the MSM algorithm using user defined seed values will each be passed to the Runs Test function for analysis. The user defined seed values will be the same and used in the same order for both algorithms. The experiment will run and the resulting random number values will be separated and grouped as such in order to make fair comparisons between the results of both random number generator algorithms.

In order to generate each list of random numbers with the different seed values for each algorithm, the time_function() is called which then calls the timeit() function to execute each algorithm. A separate empty list for each different seed value is passed so that the resulting random numbers for each algorithm and seed value are kept separate for accurate and fair data analysis. Once the time_function() is called it returns the execution time in milliseconds for each algorithm and these execution times are appended to the corresponding list holding the execution time values. There is a Middle Square Method times list and a Linear Congruential Method times list. These lists with the resulting execution times are used to generate a double bar graph to make accurate comparisons.

This study uses the Runs Test for Randomness p-value as an evaluation metric since it is a statistical test which determines if a series of values or a dataset is produced by a random process. The full name for the test is the Wald-Wolfowitz runs test. This name comes from the statisticians Abraham Wald and Jacob Wolfowitz who developed this statistical test. This test is a non-parametric statistical test used for determining the randomness of a dataset. This test uses runs of data to determine whether the dataset is produced randomly or if it follows a pattern too closely. A run can be defined as a pattern such as consecutive even or odd values or if there are consecutive numbers which are either decreasing or increasing. Another common approach is to compare the number of values above and below the mean value of the dataset, if the value is above the mean, then it is considered a positive value, and if it is below the mean value then it is a negative value. The number of runs in the dataset are counted as the first step of the Runs Test process. Next a series of calculations are used to determine the z-score test statistic and the corresponding p-value.

$$Z = \frac{\text{Observed} - \text{Expected}}{\sqrt{\text{Variance}}}$$

The z-score test statistic can be calculated by the above equation. It is the observed number of runs minus the expected number of runs divided by the square root of the variance. To calculate the expected number of runs the equation 1 + (2 * A * B / N) is used. To calculate the variance the equation

((2 * A * B * (2 * A * B – N)) / N^2 (N-1)) is used. In these equations A is the number of observations above the mean, B is the number of observations below or equal to the mean, and N is the total number of observations. To get the p-value we use the probability of observing the absolute value of a standard normal variable which is greater than the absolute value of the z-score statistic. The Runs Test for Randomness is based on deciding to either reject or accept the null hypothesis. The null hypothesis H_0 is that the data was produced in a random manner, and the alternative hypothesis is that the data was not produced in a random manner. A significance level (alpha) of 0.05 is used in this study. To determine whether to accept or reject the null hypothesis, the p-value from the resulting lists of random numbers is compared with the significance level. If the p-value is greater than or equal to alpha, then we fail to reject the null hypothesis and can confidently say the data was produced in a random manner. If the p-value falls below the significance value, there is enough evidence to say the dataset was not produced in a random manner.

The second evaluation metric used for this study is the execution time of each PRNG algorithm. Execution time was chosen since one of the main advantages of PRNGs is that they can quickly generate hundreds or thousands of random numbers. This contrasts with TRNGs which produce numbers that are more likely to be statistically random, but they may take very long periods of time to generate the same number of random numbers compared to a PRNG.

No external dataset is used for this study, but the seed values for each test run will determine or alter the resulting random numbers to be generated. This experiment uses a single seed value, and then based on the specific algorithm will generate the 50 random numbers and those results will be analyzed. In a way, this experiment generates a dataset that will be analyzed, but it will be based on the starting seed values. As mentioned previously, the user defined seed values will be noted in the results section. Based on analysis of the resulting random numbers using the stated evaluation metrics, a conclusion will be made which method of random number generation is more efficient and statistically random.

6. RESULTS ANALYSIS

This section is used to analyze the results of the experiment based on the evaluation metrics to help decide on which algorithm performs better at producing random numbers.

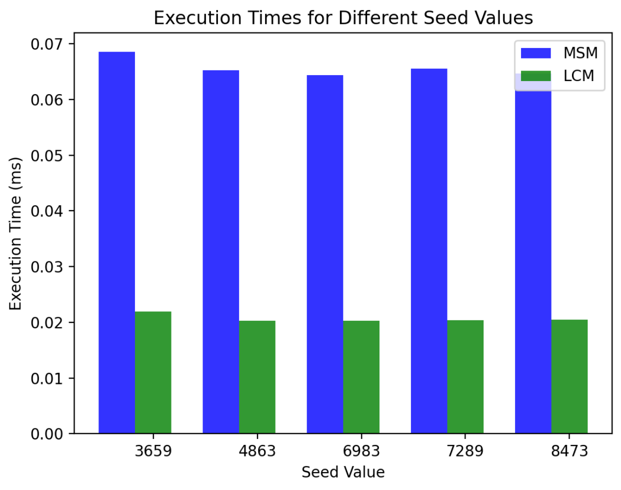

Figure 1. Execution Times of MSM and LCM

Analysis of Figure 1 shows that for each of the five different user defined seed values, the Middle Square Method took significantly longer to generate the default 50 random numbers. The Middle Square Method took more than double the time that the Linear Congruential Method took to generate the same number of random numbers. It seems that the MSM performed best in terms of execution time for seed value 6983. It appears that this seed value also performed best for the LCM. As the graph shows, the LCM far outperformed the MSM in terms of execution times in milliseconds to generate 50 random numbers.

Method	Seed	Runs Test p-value	Random? (p-value >= 0.05)
Middle Square Method	3659	0.7750513736700824	Yes (fail to reject H_0)
	4863	0.5676284992633359	Yes (fail to reject H_0)
	6983	0.48860816626622083	Yes (fail to reject H_0)
	7289	0.03079306420403246	No (reject H_0)
	8473	0.9158685490944402	Yes (fail to reject H_0)
Linear Congruential Method	3659	0.0650902667816322	Yes (fail to reject H_0)
	4863	0.5966099949082471	Yes (fail to reject H_0)
	6983	0.17837561367684152	Yes (fail to reject H_0)
	7289	0.41396562236564427	Yes (fail to reject H_0)
	8473	0.8090679300618948	Yes (fail to reject H_0)

Table 1. Runs Test for Randomness Results

Analysis of Table 1 shows that the Linear Congruential Method passed the Runs Test for Randomness for all user defined seed values. For the Middle Square Method, the seed value of 7289 produced a dataset with a p-value of 0.03 which is less than the significance level (alpha) being used of 0.05. In this case, this means that the dataset of random numbers produced can be described as not being produced in a random manner. The patterns of runs for this seed value must have been either too low or too high to produce a p-value that would satisfy the null hypothesis. All p-values for the Linear Congruential Method with each different seed satisfied the null hypothesis that the datasets were produced in a random manner. The worst performing seed value for the LCM was 3659. The p-value for this seed is at 0.06 which is the closest to the cutoff of 0.05 significance level. Based on this analysis the Linear Congruential Method outperformed the Middle Square Method using the Runs Test for Randomness p-value evaluations metric.

7. Conclusions

This study was conducted to research and experiment with multiple pseudo random number generator algorithms. Preliminary research of peer reviewed journal articles was done to provide insight and background information for the study. The chosen algorithms that were studied, implemented, and analyzed were the Middle Square Method and the Linear Congruential Method. These algorithms were chosen since they are some of the earliest known PRNG algorithms in practice. These algorithms were implemented in Python and various modules such as timeit, statsmodels, and matplotlib were used to conduct analysis of the produced datasets. The experiment conducted consisted of running each algorithm 5 different times with 5 different user defined seed values. It involved capturing the execution time for each algorithm using each of the 5 defined seed values. It also involved using the Runs Test for Randomness to conduct a statistical test for randomness on the resulting datasets.

The contribution of this study to the research of pseudo random number generators was that an unbiased experiment was set up and the algorithms were implemented using the same programming language and computer hardware. This provided for a fair and strong comparison of the different PRNG algorithms. Although the algorithms used are primitive, this study still provides great insight and detail into these algorithms for other individuals that want to learn more about PRNGs and how they can be evaluated.

Advantages for using the Middle Square Method for random number generation are that it is a simple algorithm which many individuals are capable of understanding. It is also quite simple to implement in various programming languages. A disadvantage of this algorithm is that it has a lot of processing overhead like adding padded zeros to produced numbers before the next iteration can start. Another disadvantage of this algorithm is the unpredictability of number cycles. If a cycle occurs in the process of generating the random numbers, then the same values will continue to be produced over and over. Also, if the middle four digits of the squared number are four zeros, then the rest of the random numbers in the process will be generated as zero. Moving onto the Linear Congruential Method, the advantages of this algorithm are that it is very efficient at producing the desired number of random numbers as seen in the results analysis section and the equation to calculate the random numbers is simple and can be understood by individuals with only high school level arithmetic knowledge. A disadvantage of the LCM is that the results greatly depend upon the parameters of the equation. This is not a desirable feature for PRNGs, but if the parameters are chosen correctly, this method can do a solid job at producing numbers that appear to be random.

This research is unique since two primitive pseudo random number generator algorithms were focused on to study how computers generate random number values. It can also be considered unique since the researcher developed these algorithms from scratch in the same programming language while using the same hardware specifications. It involved not only learning about how these algorithms work, but it required the implementation of the algorithms to gain much more detailed insight compared to just finding an implementation and studying the results.

Based on the analysis of the results, the Linear Congruential Method performed better for both evaluation metrics. This method took less than half of the time to generate 50 random numbers when analyzing the execution times for the 5 different seed values. Additionally, all seed values used and their respective datasets for the LCM passed the Runs Test for Randomness with a significance level of 0.05. Based on these results, the algorithm that performed the best at generating numbers which appeared to be random using the evaluation metrics of speed and a statistical test for randomness was the Linear Congruential Method.

8. REFERENCES

[1] Behnia, S., Akhavan, A., Akhshani, A., & Samsudin, A. (2011). A novel dynamic model of pseudo random number generator. Journal of Computational and Applied Mathematics, 235(12), 3455-3463. doi:10.1016/j.cam.2011.02.006

[2] Yang, YG., Zhao, QQ. Novel pseudo-random number generator based on quantum random walks. *Sci Rep* **6,** 20362 (2016). https://doi.org/10.1038/srep20362

[3] Seda Arslan Tuncer, Turgay Kaya, "True Random Number Generation from Bioelectrical and PhysicalSignals", *Computational and Mathematical Methods in Medicine*, vol. 2018, ArticleID 3579275, 11 pages, 2018. https://doi.org/10.1155/2018/3579275

[4] Anikin, I. V., & Alnajjar, K. (2018). Increasing the quality of pseudo-random number generator based on fuzzy logic. *Journal of Physics: Conference Series, 1096*, 012193. doi:10.1088/1742-6596/1096/1/012193

[5] K. Jasim Mohammad, O., Abbas, S., M. El-Horbaty, E., & M. Salem, A. (2014). A new trend of pseudo random number generation using qkd. International Journal of Computer Applications, 96(3), 13-17. doi:10.5120/16773-6

[6] A. Rotenberg. 1960. A New Pseudo-Random Number Generator. J. ACM 7, 1 (Jan. 1960), 75–77. DOI:https://doi.org/10.1145/321008.321019

[7] Xi Chen, Shuai Qian, Fei Yu, Zinan Zhang, Hui Shen, Yuanyuan Huang, Shuo Cai, Zelin Deng, Yi Li, Sichun Du, "Pseudorandom Number Generator Based on Three Kinds of Four-Wing Memristive Hyperchaotic System and Its Application in Image Encryption", Complexity, vol. 2020, Article ID 8274685, 17 pages, 2020. https://doi.org/10.1155/2020/8274685

[8] Blum L., Blum M., Shub M. (1983) Comparison of Two Pseudo-Random Number Generators. In: Chaum D., Rivest R.L., Sherman A.T. (eds) Advances in Cryptology. Springer, Boston, MA. https://doi.org/10.1007/978-1-4757-0602-4_6