Informatics and Control Problems 45 Issue 2 (2025)
journal homepage: www.icp.az

A function generating pseudorandom numbers with a uniform distribution for creating cryptographic keys

Elkhan Sabziev, Adalat Pashayev, Emin Babayev[*]

Institute of Control Systems, Baku, Azerbaijan

ARTICLE INFO

Article history:
Received 12.03.2025
Received in revised form 29.03.2025
Accepted 07.04.2025
Available online 26.12.2025

Keywords:
Pseudorandom number generators
Perfect secrecy
Discrete uniform distribution
Generation of cryptographic keys

ABSTRACT

The security of cryptographic keys depends on the distribution and unpredictability of the generated random numbers. Therefore, it is required that the generated pseudorandom number sequence has a distribution close to a discrete uniform distribution. In this study, an algorithm for generating pseudorandom number sequences with a discrete uniform distribution is proposed.

1. Introduction

Systems that perform data transformations using keys, encryption and decryption algorithms to ensure data confidentiality are called cryptographic systems [1]. According to Kerckhoffs' principle, the security of a cryptographic system should be based on the level of the key secrecy, even if all information about the system, including the encryption algorithm, is known. [2]. In cryptographic systems, deriving a long key from a small password allows this key to be used securely for cryptographic operations. The predictability and non-randomness of the generated keys can make the system vulnerable to cryptanalytic attacks. Generating a random sequence of numbers is one of the main elements used in key generation [1]. Pseudorandom number generators (PRNGs) are software tools that generate random sequences of numbers using a specific mathematical algorithm and an initial input parameter called a seed [1]. PRNGs are widely used in cryptographic systems for various purposes. In this regard, the creation of random number sequence generators is important.

In systems with perfect secrecy, as defined by Claude Shannon in 1949, there is no information leakage between the encrypted message and the original message [3]. This means that obtaining the encrypted message does not provide any information about the original plaintext message. If we denote the a priori probability of message M by $P(M)$, and the a posteriori probability of message M, if encrypted message E is intercepted, by $P_E(M)$, then for perfect secrecy the equality $P_E(M) = P(M)$ must be satisfied. [3]. According to Shannon's theorem, a necessary and sufficient condition for perfect secrecy is that, for all possible messages and encrypted messages, the probability of the encrypted message being intercepted does not depend on the original message. In other words, given

[*]Corresponding author
E-mail addresses: elkhan.sabziev@gmail.com (E.N. Sabziev), adalat.pashayev@gmail.com (A.B. Pashayev), eminbabayev2025@gmail.com (E.M. Babayev)

www.icp.az/2025/2-05.pdf https://doi.org/10.54381/icp.2025.2.05
2664-2085/ © 2025 Institute of Control Systems. All rights reserved

any message M, the probability of the cryptogram E being intercepted must be equal to the probability of E being intercepted from any cause, mathematically this condition is expressed as follows:

$$P_M(E) = P(E),$$

where M is the original message, E is the encrypted message, $P_M(E)$ is conditional probability of cryptogram E if message M is chosen, while $P(E)$ is the probability of probability of obtaining cryptogram E from any cause [3].

The correct choice of keys is one of the main issues for achieving perfect secrecy. For perfect secrecy, the length of the key should be equal to the length of the message to be encrypted, and the symbols of the key should be distributed uniformly [3]. The non-equal probability of symbols causes some symbols to be used more often than others. In this case, such weaknesses in the keys make the cryptographic system vulnerable to attacks. The random and equally probable selection of the symbols of the keys prevents key predictability and increases the stability of the cryptographic system.

The amount of uncertainty in a given key is measured by the entropy formula, proposed in 1948 by Claude Shannon [4]. Entropy is a measure of uncertainty and is calculated as follows:

$$H = -\sum_{i=1}^{n} P_i \log P_i,$$

where P_i is the probability of state i. For n possible outcomes, H takes a maximum value if the probability of each outcome occurring is $P_i = \frac{1}{n}$. In this case, the entropy is equal to $\log n$ [4].

2. Overview of the research on the topic

Pseudo-Random Number Generators (PRNGs) are deterministic algorithms that generate sequences of numbers that appear random in a computer environment [1]. In fact, given that computers are deterministic devices, it is impossible to say that the random numbers generated in this way are completely random. Hence the term "pseudo" is used. When started with the same seed value as an input parameter, PRNGs execute the same steps of the algorithm, and thus the sequence of random numbers produced by the generator is always the same. This property is used in the generation of cryptographic keys.

In cryptographic applications, TRNGs (True Random Number Generators) and CSPRNGs (Cryptographically Secure Pseudorandom Number Generators) based on complex cryptographic algorithms are used to generate keys. TRNGs use non-deterministic sources, such as natural physical phenomena, to generate randomness. The fact that the results obtained are truly random allows TRNGs to ensure a high level of security for cryptographic purposes [1]. However, TRNGs are slower than CSPRNGs, meaning they take longer to generate numbers. For this reason, their use in some cryptographic systems may be inefficient. For instance, in real-time data exchange or when large amounts of data need to be encrypted quickly, CSPRNGs are more appropriate because they are faster. At the same time, TRNGs are dependent on specific hardware modules (e.g., HSM – Hardware Security Module, TPM – Trusted Platform Module, or specific TRNG chips), so their use is not practical for every cryptographic system. For this reason, CSPRNGs are more widely used. The results of such random number generators are expected to have a uniform distribution [1]. This property is necessary to ensure randomness. Weak PRNGs can result in some numbers being generated more often than others, which reduces randomness. CSPRNGs have more complex algorithms than other PRNGs. Despite their deterministic nature, CSPRNGs should be difficult to predict. The Blum Blum Shub, Yarrow, ChaCha20, and Fortuna algorithms are examples of CSPRNGs.

The Blum Blum Shub (BBS) algorithm is a PRNG proposed in 1986 by Lenore Blum, Manuel Blum, and Michael Shub [5]. Let us look at Blum Blum Shub generator in more detail. The algorithm of the $x^2 \text{mod } N$ generator performs the following steps

1. Parameter selection:
   a) Two large, distinct prime numbers of equal length, p and q, are selected. For both primes, $p \equiv 3 \; mod \; 4$ and $q \equiv 3 \; mod \; 4$.
   b) The parameter N is calculated by the product of these prime numbers:
$$N = q \times p.$$
2. Selection of the seed value, which is the initial input parameter:
   a) This value is denoted by s and is used to create the element x_0 in the first step of the algorithm. x_0 must be a prime number different from 0 and 1 and coprime with N. That is, p and q must not be divisors of s. Using the seed value s, x_0 is found as follows:
$$x_0 = s^2 \; mod \; N.$$
3. Random number generation:
   a) The algorithm repeats the following process to generate each new random number:
$$x_{n+1} = {x_n}^2 \; mod \; N.$$
   b) The random bits are obtained by subtracting some bits from the value x_n obtained at each step. In general, only the least significant bit (LSB) is selected and used. The least significant bit (LSB) is the rightmost bit in the binary representation of a number. [5]

3. Problem statement

It is required to find an algorithm that generates a sequence of pseudorandom numbers such that the distribution of the elements of this sequence is close to a uniform distribution.

Let us assume that the set of possible values of a discrete random variable X is a proper set $S = \{x_1, x_2, \dots, x_n, \dots\}$. The function that determines the probability of X being equal to a certain value of x is called the probability mass function (PMF) and is determined as follows:
$$p_X(x) = P(X = x) \,,$$
where $x \in S$ [6]. Note that for all possible values of X, $p_X(x) \geq 0$, $\sum_{x \in S} p_X(x) = 1$ [7].

Definition 1 (Discrete uniform distribution). For a given set of finite numbers $C = \{x_1, x_2, \dots, x_n\}$, a random variable X that selects each element from this set with equal probability is called a discrete uniform distribution:
$$P(X = x_i) = \frac{1}{n} \,,$$
where $x_i \in C$ and $n = |C|$ [6].

A discrete uniform distribution over a set of finite numbers is a distribution in which every possible number is equally probable.

4. Proposed approach

To solve the problem, we consider the distribution of the values of the function $f_n = \sin(\lambda n) \,, n = 1,2, \dots, N$ in the interval $[-1,1]$. Here, the parameter λ denotes the angle, in degrees. The domain of the function f_n is a set of all real numbers, $D(f_n) = R = (-\infty, +\infty)$, where the set D indicates the domain of the function f_n. The values that a function can take vary in the interval $[-1,1]$ and the set of values of the function is denoted by E, $E(f_n) = [-1,1]$. To check the uniform distribution of the function in the interval $[-1,1]$, we will divide $[-1,1]$ into m equal parts. We will denote the resulting fragments by b_i and see how many values of the function f_n fall into each fragment. The fragments b_i are generally determined as follows:

$$b_i = [-1 + \Delta h \times i, -1 + \Delta h \times (i+1)], \tag{1}$$

where $i = 0,1, \dots, m-1$, obviously, each fragment b_i is determined within $[-1,1]$. Δh is the length of fragments b_i, and $\Delta h = \frac{2}{m}$.

It can be seen from the result of the numerical experiments that when the values of the parameter n of the function f_n increase in discrete steps in the interval $(0, +\infty)$, the number of values of the function is not distributed uniformly in $[-1; 1]$. The number of values of the function is smaller in the middle of interval $[-1; 1]$, and while it is distributed close to the uniform distribution, the number of values of the function is larger at the edges of the interval (see Fig. 1).

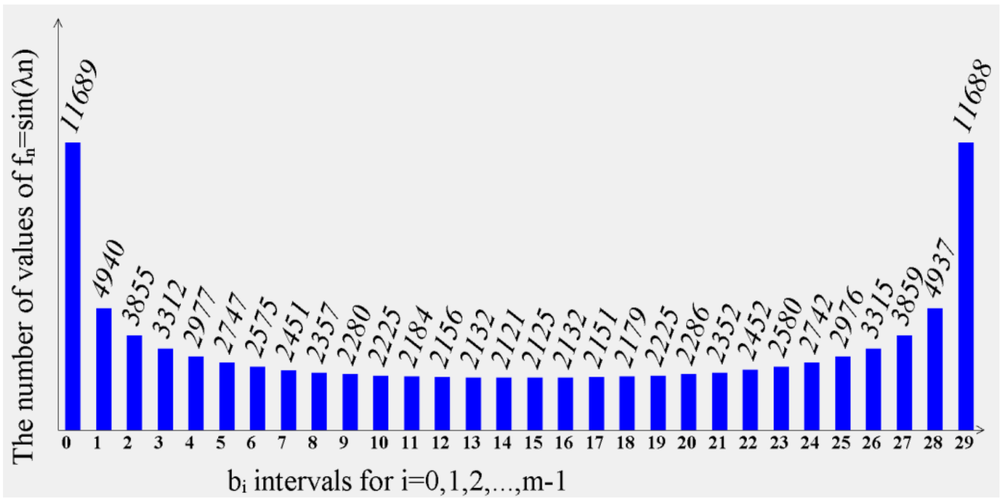

Fig. 1. Result of the experiment for $\lambda = 3.1, n = 100000$, and $m = 30$.

In this study, two classes of functions serving the same purpose will be denoted by ${f_n}^{(1)}$ and ${f_n}^{(2)}$, respectively. Based on the above, we will modify the sine function as follows:

$${f_n}^{(1)} = \frac{\sin(\lambda n)}{\alpha}, \tag{2}$$

where $0 < \alpha < 1$, $n = 1,2, \dots, N$, obviously, ${f_n}^{(1)} \in \left[\frac{-1}{\alpha}, \frac{1}{\alpha}\right]$. Let us introduce the function in the following form:

$${f_n}^{(2)} = \begin{cases} {f_n}^{(1)} & \text{if} \quad \left|{f_n}^{(1)}\right| \le 1, \\ 1 - {f_n}^{(1)} & \text{if} \quad {f_n}^{(1)} > 1, \\ -1 - {f_n}^{(1)} & \text{if} \quad {f_n}^{(1)} < -1. \end{cases} \tag{3}$$

The elements of the random number sequence generated by the proposed function do not coincide.

Theorem. If λ and α are rational numbers, then the elements of the sequence ${f_n}^{(2)}$ do not coincide for different values of n.

Proof. Suppose that when $m \neq n$, ${f_n}^{(1)} = {f_m}^{(1)}$, since ${f_n}^{(1)} = \frac{\sin(\lambda n)}{\alpha}$, ${f_m}^{(1)} = \frac{\sin(\lambda m)}{\alpha}$,

$$\sin(\lambda n) = \sin(\lambda m),$$

$$\sin(\lambda n) - \sin(\lambda m) = 0. \tag{4}$$

Using the formula for converting the difference of trigonometric functions into a product [8], we can write equation (4) as follows:

$$\sin(\alpha) - \sin(\beta) = 2\sin\left(\frac{\alpha-\beta}{2}\right)\cos\left(\frac{\alpha+\beta}{2}\right),$$

$$\sin(\lambda n) - \sin(\lambda m) = 2\sin\left(\frac{\lambda n-\lambda m}{2}\right)\cos\left(\frac{\lambda n+\lambda m}{2}\right) = 0. \qquad (5)$$

If λ is a rational number, then the numbers $\frac{\lambda n-\lambda m}{2}$ and $\frac{\lambda n+\lambda m}{2}$ in equation (5) are also rational. The sine and cosine functions do not take the value 0 for any rational value of the argument. Thus, when λ is a rational number, the equality ⟦$f_n^{(1)} = f_m^{(1)}$ is not satisfied for different values of m and n, and the values of the functions ⟦$f_n^{(1)}$, $f_m^{(1)}$ do not coincide. Thus, since the values of the function $f_n^{(1)}$ do not repeat, the elements of the sequence $f_n^{(2)}$ do not repeat either. The theorem is proved.

Let us consider the problem of a uniform distribution of the sequence $f_n^{(2)}$ in interval $[-1; 1]$. Similarly, let us consider how many elements of the sequence $f_n^{(2)}$ fall into each fragment b_i. Note that the elements of $f_n^{(2)}$ that do not fall into interval $[-1; 1]$ are discarded. In order to verify that the sequence $f_n^{(2)}$ generates random numbers close to a uniform distribution, appropriate software was created and numerical experiments were conducted. The results of these experiments, along with their analysis and relevant graphical representations, are presented separately in Section 6.

5. Information about the software created for the purpose of the experiment

The algorithm used in the software generates a pseudo-random number sequence using functions (2) and (3) depending on the parameters entered by the user. It is checked whether the values of the sequence $f_n^{(2)}$ have a uniform distribution in interval $[-1,1]$. Based on the input parameters, $[-1,1]$ is divided into equal-length sections and the number of values of the sequence $f_n^{(2)}$ falling in each section is viewed. The result is displayed in the form of a graph. The software is written in the Java programming language. The algorithm consists of the following steps:

1. Obtaining initial input parameters: The input parameters of the algorithm are the values of α, λ and N, which are the parameters of the function (2), and the value of m, which indicates the number of parts into which interval $[-1,1]$ will be divided, entered by the user. These values constitute the main parameters of the algorithm and are assigned to the variables shown below. Main variables used in the software:

- $\boldsymbol{alfa}$**,** *double*-type variable, takes a value between 0 and 1.
- $\boldsymbol{lambda}$**,** *double*-type variable, used to assign the angle, which is a parameter of the sine function, given in degrees.
- $\boldsymbol{N}$**,** *int*-type variable, the value of the variable indicates how many times the calculations will be repeated.
- $\boldsymbol{m}$**,** *int*-type variable, the value of the variable indicates the number of subintervals into which interval $[-1,1]$ will be divided.

2. Calculating the length of subintervals: The algorithm divides interval $[-1,1]$ into m equal sections. To calculate the length of the sections, *double*-type variable $\boldsymbol{deltaH}$ is introduced and the length of the sections is calculated as $deltaH = 2/m$.

3. Determining the $\boldsymbol{count}$ array: An *int*-type array $count$ with m elements is created and each element of the array is initially set to zero. The elements of this array will store the number of values of $f_n^{(2)}$ that fall into each of the intervals $b_i, i = 0,1,2,\dots,m-1$, as defined by (1).

4. Main cycle (repeats $\boldsymbol{N}$ times, $\boldsymbol{n = 1, 2, 3, \dots, N}$): The algorithm calculates the values of the sequence $f_n^{(2)}$ in each iteration of the N-times running cycle based on the initial input parameters entered by the user, and increments the values of the elements of the $count$ array based on the interval in which those values fall. The following operations are performed in each iteration:

- $\boldsymbol{sinusValue}$ *double*-type variable is introduced, and the value of $\sin(\lambda n)$ function is calculated and assigned to this variable.
- $\boldsymbol{fn1Value}$ *double*-type variable is introduced, and the value of the sine function is divided by the value of the $alfa$ variable and assigned to this variable.
- **Determining the values of the sequence $\boldsymbol{f_n}^{(2)}$ i: $\boldsymbol{fn2Value}$** *double*-type variable is introduced. Using the value of the variable $fn1Value$, the value of the variable $fn2Value$ is determined within the following conditions:
    - If $|fn1Value| \leq 1$:
      $$fn2Value = fn1Value$$
    - If $fn1Value > 1$:
      $$fn2Value = 1 - fn1Value$$
    - If $fn1Value < -1$:
      $$fn2Value = -1 - fn1Value$$
- **Checking into which interval the values of the sequence $\boldsymbol{f_n}^{(2)}$ fall:** in this step, the values of the sequence $f_n^{(2)}$ that do not fall within the $[-1,1]$ segment are ignored. If the value of the variable $fn2Value$ belongs to the interval $[-1,1]$, the following operations are performed:
- **Internal cycle (repeats $\boldsymbol{m}$ times, $\boldsymbol{i = 0, 1, 2, \dots, m-1}$):** the following operations are performed in each iteration:
- $\boldsymbol{p}$ and $\boldsymbol{q}$ *double*-type variables are introduced. The values of the variables p and q determine the boundaries of the fragment b_i for each i and are calculated as $p = -1 + \Delta h \times i$, $q = -1 + \Delta h \times (i + 1)$.
- If $p \leq fn2Value \leq q$, the corresponding element of the $count$ array is incremented by 1 and the cycle is stopped. The operation is repeated for the next value of $f_n^{(2)}$.

5. Displaying the result: Using the $count$ array, the number of the values of $f_n^{(2)}$ falling into each b_i fragment is printed in histogram form.

6. Results of the experiment

It is assumed that the values of the parameters α and λ should be selected so that the distribution of the sequence $f_n^{(2)}$ is close to a uniform distribution. The experiment shows that for values of the parameter α close to zero (e.g., $\alpha \approx 0.15 \pm 0.05$), the sequences obtained are close to a uniform distribution. For larger values of the parameter α, the values of the sequence obtained are not uniformly distributed. For comparison, the results corresponding to the values $\alpha = 0.2$, $\alpha = 0.3$ and $\alpha = 0.9$ are shown in Fig. 2, 3 and 4.

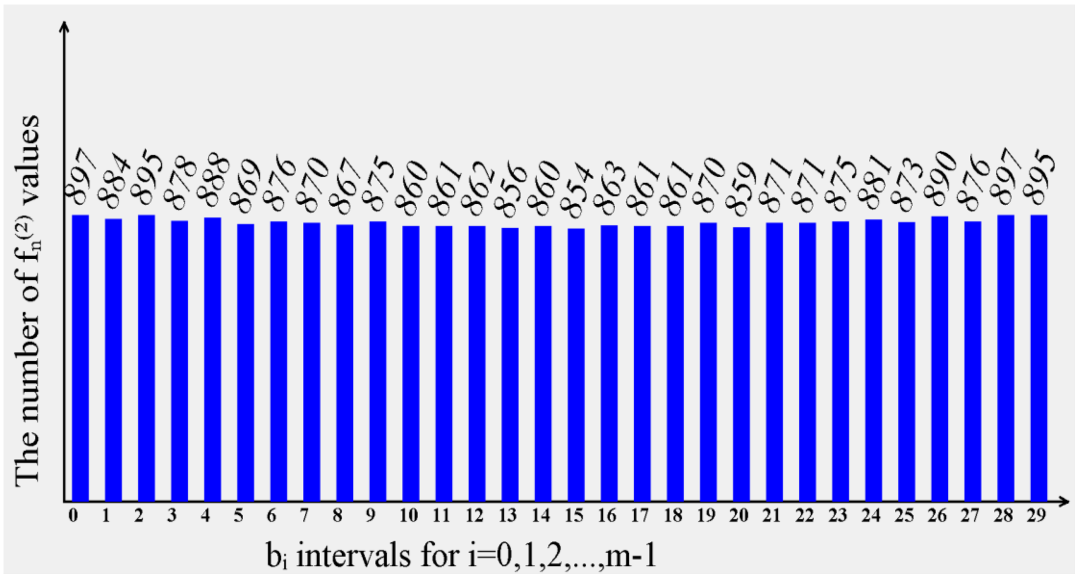

Fig. 2. Result of the experiment for $\alpha = 0.2, n = 100000, \lambda = 3.1$ *and* $m = 30$.

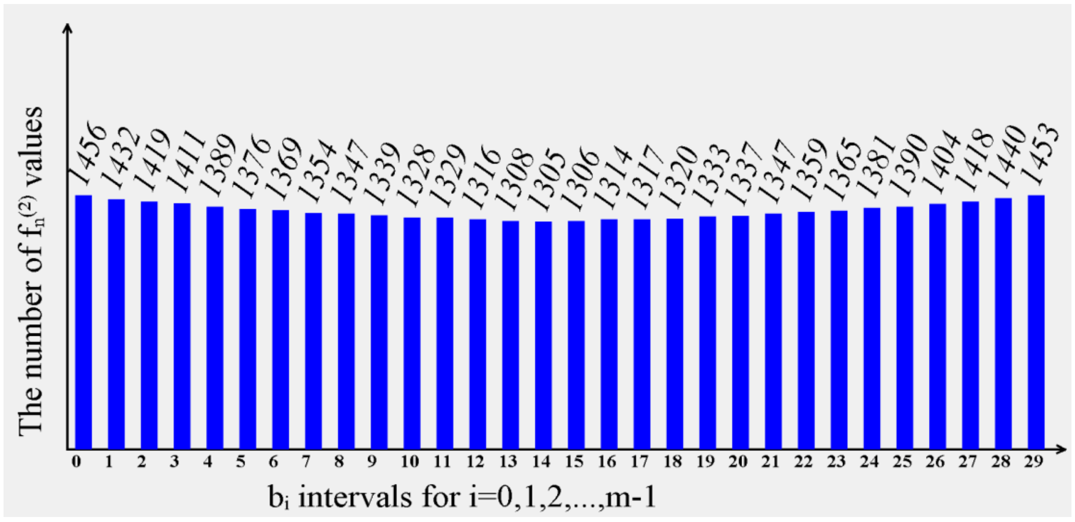

Fig. 3. Result of the experiment for $\alpha = 0.3, n = 100000, \lambda = 3.1$ *and* $m = 30$.

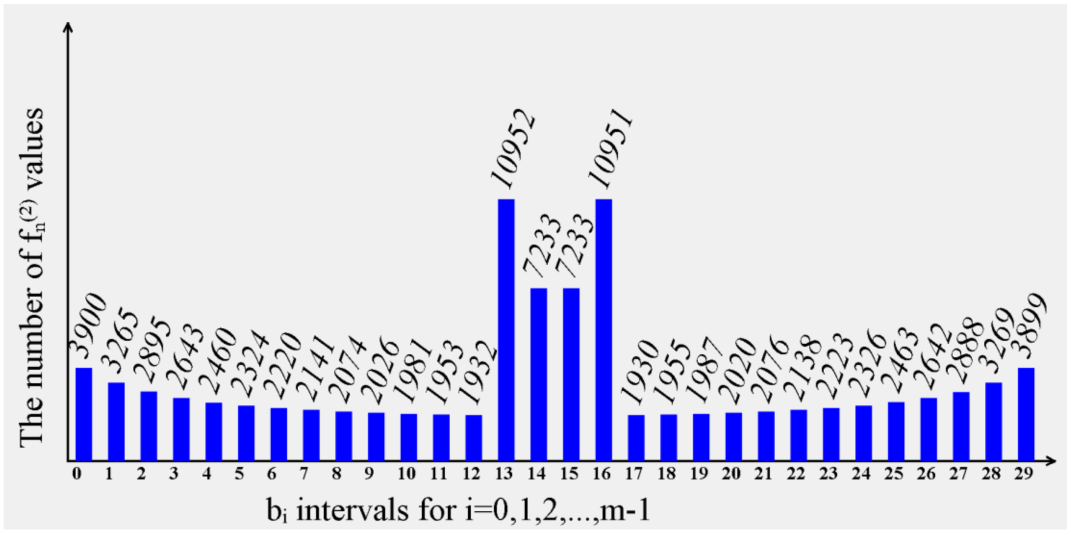

Fig. 4. Result of the experiment for $\alpha = 0.9, n = 100000, \lambda = 3.1$ *and* $m = 30$.

7. Conclusion

The study examines the numerical sequence generated by the sine function with a certain coefficient. It is shown through numerical experiments that the proposed function generates a pseudorandom sequence of numbers with a uniform distribution. In cryptographic systems with perfect secrecy proposed by Claude Shannon, deriving a long key with perfect secrecy from a small password can be ensured by randomly selecting the symbols of the key with a uniform distribution. Since these sequences have the properties of randomness and uniform distribution, they can be used to generate cryptographic keys with perfect secrecy.

References

[1] W. Stallings, Cryptography and network security: Principles and practice, 7th ed., Pearson, (2017) 753 p.

[2] J. Katz, Y. Lindell, Introduction to modern cryptography, 3rd ed., CRC Press, Taylor & Francis Group, (2021) 626 p.

[3] C.E. Shannon, Communication theory of secrecy systems, The Bell System Technical Journal, October 1949. 28 No.4 p.656-715. https://doi.org/10.1002/j.1538-7305.1949.tb00928.x

[4] C.E. Shannon, A mathematical theory of communication, The Bell System Technical Journal, July 1948. 27 No.3 p.379-423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[5] L. Blum, M. Blum, M. Shub, A simple unpredictable pseudo-random number generator, SIAM Journal on Computing. 15 No.2 (1986) p.364-383. https://doi.org/10.1137/0215025

[6] J.K. Blitzstein, J. Hwang, Introduction to probability, 1st ed., CRC Press, Taylor & Francis Group, (2015) 570 p.

[7] S.M. Ross, A first course in probability, 10th ed., Pearson, (2018) 848 p.

[8] M.İ. Əfəndiyev, Triqonometrik tənliklər, M.İ. Əfəndiyev. N.B. Kərimov, N.İ. Mahmudov, Bakı, Təbib Nəşriyyatı, (1992) 148 p. [In Azerbaijani: M.I. Efendiyev, Trigonometric Equations, M.I. Efendiyev. N.B. Kerimov, N.I. Mahmudov, Baku, Tabib Publishing House].