Pseudorandom sequence generator based on the computation of ln 2

Ivan Opirskyy[1,*,†], Oleh Harasymchuk[1,†], Olha Mykhaylova[1,†], Oleksii Hrushkovskyi[1,†] and Pavlo Kozak[1,†]

[1] *Lviv Polytechnic National University, 12 Stepana Bandery str., 79000 Lviv, Ukraine*

Abstract

This paper discusses creating a pseudorandom sequence generator using the natural logarithm of the number 2 (ln 2) calculator. Pseudorandom sequence generators are key elements in cryptography, modeling, and numerical methods, where high-quality randomness is required. Traditionally, various mathematical algorithms are used for this purpose, but we propose a new approach based on the numerical properties of ln 2. The paper describes in detail the method of computing ln 2 using the Taylor series and demonstrates how these calculations can be integrated into a pseudorandom sequence generator. The main idea is to use the ln 2 approximation to initialize the generator, allowing for the creation of number sequences with a high degree of randomness. The use of ln 2, known for its mathematical stability and accuracy, opens new horizons for generating numbers that are important for many scientific and engineering applications. The presented test results show that the proposed method provides uniform distribution and passes the standard NIST statistical tests. This demonstrates the potential of using mathematical constants and their numerical computations to improve the characteristics of pseudorandom sequence generators. Our approach offers the possibility of creating generators with improved characteristics without significantly increasing computational complexity. Additionally, we discuss potential directions for improving the generator, including optimizing the algorithm and expanding to other mathematical constants. This approach not only enhances the quality of pseudorandom sequences but also provides new tools for research in number theory and computational mathematics. An important aspect is that the proposed method provides high generation speed, making it attractive for use in real-world applications where computation time is a critical parameter. Thus, our generator may find wide application in various fields, including cryptographic protocols, simulation algorithms, and other numerical methods that require high-quality randomness and computational efficiency.

Keywords

pseudorandom sequence generator, pseudorandom number generators, mathematical algorithms, Taylor series, NIST statistical tests, mathematical constants, cryptographic protocols, simulation algorithms, computational efficiency

1. Introduction

Pseudorandom Number Generators (PRNGs) and Pseudorandom Sequence Generators (PSGs) are key elements in many scientific and technical fields. They play a crucial role in modern technologies, providing the basis for numerous applications in computer science, cryptography, statistical sampling, modeling, and simulations [1–7]. These generators enable the creation of number sequences that, while deterministic, appear random, which is critically important for ensuring data security, model accuracy, and algorithm reliability. In a world where information is becoming increasingly valuable, understanding and using PRNGs and PSGs is essential for developing effective solutions in various fields, from finance to gaming. Thus, the importance of pseudorandom number generators is hard to overestimate, as they provide the foundation for innovation and development in many sectors [8]. Particularly noteworthy is their importance in cybersecurity, where they are also a key element, and are used in solving various tasks, namely for data encryption [9], authentication [10, 11], key generation, digital signature creation algorithms, and in testing and evaluating security [12–18]. Therefore, developers of such generators face high demands for the quality of the output sequences: unpredictability, statistical independence, cryptographic robustness, and maximum generation speed. Ensuring a high quality of randomness is an important task, as it affects the reliability and accuracy of many algorithms and systems where the generated sequences will be applied. Traditionally, various mathematical algorithms and methods are used to generate pseudorandom numbers,

CQPC-2024: Classic, Quantum, and Post-Quantum Cryptography, August 6, 2024, Kyiv, Ukraine

* Corresponding author.
† These authors contributed equally.
ivan.r.opirskyi@lpnu.ua (I. Opirskyi); garasymchuk@ukr.net (O. Harasymchuk); mykhaylovaolga1@gmail.com (O. Mykhaylova); oleksii.hrushkovskyi.kb.2022@lpnu.ua (O. Hrushkovskyi); pavlo.kozak.kb.2022@lpnu.ua (P. Kozak)

0000-0002-8461-8996 (I. Opirskyi); 0000-0002-8742-8872 (O. Harasymchuk); 0000-0002-3086-3160 (O. Mykhaylova); 0009-0007-3626-9780 (O. Hrushkovskyi); 0009-0005-0432-4541 (P. Kozak)

including the linear congruential method [19–21], shift register generators [22–24], the Lagrange method [25], Mersenne Twister algorithms [26], and others [27–30]. However, existing approaches do not always meet the requirements for uniform distribution and passing statistical tests, so researchers continuously search for new methods, ways, and algorithms to generate pseudorandom sequences that meet the growing demands for their quality.

2. Problem statement

The research problem involves seeking a new approach to generating pseudorandom numbers that would ensure high-quality randomness and computational efficiency. One such approach is the use of the numerical properties of mathematical constants [31]. The natural logarithm of number 2 (ln 2) has a range of unique properties that make it promising for use in generating pseudorandom numbers. Existing studies demonstrate the effectiveness of using logarithms of mathematical constants in cryptography and numerical methods [32–34], but the integration of ln 2 into pseudorandom number generators remains insufficiently explored.

This paper aims to develop and test a pseudorandom number generator based on the computation of ln 2 using the Taylor series. The research tasks include describing the methodology for computing ln 2, integrating these computations into a pseudorandom number generator, conducting testing, assessing the quality of the generated sequences, and analyzing the results.

Lastly, this solution offers robust change and feature management capabilities. This means that it can easily adapt to evolving business needs, with the ability to incorporate new features and make necessary changes in a timely and efficient manner. This flexibility ensures the solution remains relevant and continues to deliver value over time [35].

3. Research analysis

Historically, the concept of logarithms was introduced in the 17th century by the Scottish mathematician John Napier, who first developed logarithmic tables [36]. His work significantly simplified the process of multiplying and dividing large numbers, which was extremely useful for astronomy, navigation, and other sciences. Natural logarithms, based on the number e (approximately equal to 2.71828), appeared later and became important in mathematical analysis thanks to the works of Leibniz and Euler [37, 38].

One of the classic methods for computing e is using the Taylor series. For example, one can use the expansion of the logarithmic function into a Taylor series around 1:

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots . \quad (1)$$

For x = 1, we get:

$$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots . \quad (2)$$

This series converges relatively slowly, so more efficient methods are typically used for practical calculations [39]. Another approach is based on numerical integration methods. The natural logarithm can be defined as a definite integral:

$$\ln 2 = \int_1^2 \frac{1}{x}\, dx . \quad (3)$$

For numerical computation of this integral, methods such as the rectangle (midpoint), trapezoidal, or Simpson's rule are applied, which allows for obtaining more precise values [40].

One of the most powerful methods of computing

Ln 2 is the Newton-Raphson method, which is used for finding the roots of equations and can be adapted for computing logarithms [41]. Starting with the equation $e^y = 2$, where $y = \ln 2$, we can formulate a function $f(x) = e^x - 2$ and apply the Newton-Raphson method to solve for

$$x_{n+1} = x_n - \frac{e^{x_n} - 2}{e^{x_n}} = x_n - 1 + \frac{2}{e^{x_n}} . \quad (4)$$

A clear downside is that the use of Euler's constant e for computation is required, which itself is transcendental and cannot be precisely calculated. Therefore, it is necessary to calculate the constant itself simultaneously, which increases the number of operations. On the other hand, in such a case, one iteration can generate a much larger sequence of numbers than other algorithms.

With the advent of computers, an obvious application became the computation of mathematical constants. In one of the first works on this topic [42], the following formula was used to compute ln 2:

$$\ln 2 = \sum_{k=1}^{\infty} \frac{1}{k * 2^k} . \quad (5)$$

This represents a series of transformations over the classical Taylor series expansion [43]. In 1995, there was an unprecedented breakthrough in the calculation of constants. French mathematician Simon Plouffe discovered a series that allowed the computation of the i^{th} hexadecimal digit of π [44]. The formula is as follows:

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right] . \quad (6)$$

Later, other variations of this formula were found for other constants [35], among which we are, of course, interested in ln(2), which is represented by the formula:

$$\ln 2 = \frac{1}{2} \sum_{k=0}^{\infty} \frac{1}{2^k} \frac{1}{k+1} . \quad (7)$$

Also, a representative of this type of formula is the aforementioned formula (5). These formulas have allowed for the effective calculation of constants starting from any hexadecimal number [45], ensuring low resource consumption, but in practice, the approximation to the exact result occurs slowly. Some formulas ensure the accuracy of calculations at the expense of using more computer

resources. For example, in 1997 a record of 10,079,926 digits was set [46], which compared to the results of 1962 [42], i.e., 3863 digits, is a significant breakthrough. This was achieved using the Mercator series [47], and in this case, by the formula:

$$\ln 2 = 2\mathrm{L}(5) + 2\mathrm{L}(7), \quad (8)$$

where

$$\mathrm{L}(k) = arctanh\frac{1}{k} = \frac{1}{2}\ln\left(\frac{k+1}{k-1}\right). \quad (9)$$

Over time, the basic principle of calculation has not changed, as the use of such formulas allows obtaining precise values, although it requires a large amount of computation. For example, the record for the number of digits after the decimal was set in 2021 by William Ekols [48] and represents $1.5 \cdot 10^{12}$ digits after the comma. To calculate such several digits, it took 98.9 days [49], and for the correctness check—61.7. And this is on a machine that has 48 cores and 256 GB of RAM.

Also, do not forget the software that was used to calculate such several digits. This software is called y-cruncher and was created in 2009 [50]. The main advantage of this software is its optimization and maximum efficiency in resource use thanks to various techniques [51]. For the calculation, this formula is used:

$$\ln 2 = 18\mathrm{L}\left(\frac{1}{26}\right) - 2\mathrm{L}\left(\frac{1}{4801}\right) + 8\mathrm{L}\left(\frac{1}{8749}\right). \quad (10)$$

The most recent record as of now for computing the number of digits after the decimal is $3 \cdot 10^{12}$ digits after the comma, set by Jordan Ranous on February 12, 2024 [52]. A machine with 2 × Intel Xeon Platinum 8460H (a total of 80 cores) and 512 GB of RAM was used. The computation took 42.7 days, while the correctness check took 58.3 days.

In conclusion, it can be noted that in calculating ln(2), we face two extremes: formulas that allow efficiently, in terms of resource use, to calculate this constant but for accuracy lose their speed or those that use a large number of resources. Ideally, finding a compromise would be optimal, but when we face the task of precise calculation of ln(2), this option does not exist. Accordingly, by shifting the focus from precise calculation of ln(2) to using knowledge about this constant for generating pseudorandom sequences, we can achieve a compromise and obtain the desired result.

In this paper, we will take a detailed look at using the Taylor series for generating pseudorandom sequences and demonstrate an algorithm that allows efficiently obtaining binary random sequences of great length that pass NIST statistical tests through which the quality of the generated pseudorandom sequence can be best assessed [53–57].

4. The main part

4.1. Analysis of the Taylor series for ln 2

As mentioned earlier, the constant ln(2) is decomposed into the following Taylor series:

$$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots. \quad (11)$$

After performing a series of operations, we obtain such a definition of the series:

$$\ln 2 = \frac{1}{1*2} + \frac{1}{3*4} + \frac{1}{5*6} + \cdots. \quad (12)$$

That has the general form:

$$\ln 2 = \sum_{n=1}^{\infty} \frac{1}{(2n-1)2n}. \quad (13)$$

Let's move on to the binary representation of ln 2. In essence, it is:

$$\ln 2 = \sum_{i=1}^{\infty} \frac{1}{2^i} x_i, \quad (14)$$

where x_i is the value of the i^{th} bits.

However, it should be noted here that we are not interested in calculating the exact value in its mathematical essence, we are interested in the non-periodicity of the bit sequence, what it contains, respectively, the operations of multiplication/division by 2, offsets, adjustments at a random place in the sequence, etc. affect the non-periodicity of the sequence and can be used.

Accordingly, the problem can be reduced to finding a way to obtain a sequence of bits from (13).

Let's consider one way to solve this problem:

The binary number system includes only two values: 0 and 1, respectively, they can be used for logical "Yes" or "No".

The proposed method determines whether a specific iteration of formula (13) is in the interval $[x_i, x_{i+1}]$ and localizes this interval. In it is the principle, it resembles a mixture of Newton's method and arithmetic coding.

4.2. Development and improvement of the algorithm for the generation of pseudorandom sequences based on the calculation of the Taylor series

The operation of the algorithm can be represented as follows:

Input: sequence_length
n: = 1
numerator_buffer = 0
denominator_buffer = 1
sequence[sequence_length]
Repeat sequence_length times
sequence_item = n * (n+1)
numerator_buffer = numerator_buffer *
sequence_item+denominator_buffer
denominator_buffer * = sequence_item

If (numerator_buffer "1" >= denominator_buffer:
sequence[i-th] = 1
numerator_buffer = (numerator_buffer "1" -
denominator_buffer
If numerator_buffer == 0:
denominator_buffer = 1

Otherwise:
sequence[i-th] = 0
numerator_buffer " = 1
n+ = 2
Output: sequence

Let's consider this algorithm in more detail. Let's assume that a segment of length is given. We iterate the series. We check whether the iteration value belongs to the interval [0.5, 1]. If it does, one is entered into the sequence, and the segment is localized by subtracting the doubled numerator from the denominator. If it is not, then zero is entered into the sequence, and the segment is localized by doubling the numerator.

Overall, this method does not ensure the exact calculation of any type of Taylor series, but due to its intricate structure, it can be used as a basis for generating pseudorandom numbers. This is because it involves a series of manipulations with non-periodic constants, which empirically should enable the generation of pseudorandom sequences based on them. Additionally, the technical aspect of the algorithm's operation, which includes the overflow of some variables, should not be overlooked. Although this deprives us of calculation precision, it introduces a certain randomness. To verify the quality of the algorithm, let's analyze the results of testing the obtained sequences using series (3) with the NIST statistical test suite:

Table 1
Test results of the algorithm for generating pseudorandom sequences based on the calculation of the Taylor series using the NIST statistical test package

Statistical Test	p-value	Pass Rate	Status
Frequency	0.000199	7/10	Failed
Block Frequency	0.000000	10/10	Failed
Cumulative Sums	0.008879	8/10	Pass
Runs	0.000000	0/10	Failed
Longest Run	0.000000	1/10	Failed
Rank	0.000000	0/10	Failed
FFT	0.000000	0/10	Failed
Non-Overlapping Template	0.000000	0/10	Failed
Overlapping Template	0.000000	2/10	Failed
Universal	0.017912	7/10	Failed
Approximate Entropy	0.000000	0/10	Failed
Random Excursions	–	3/3	Pass
Random Excursions Variant	–	3/3	Pass
Serial	0.066882	9/10	Pass
Linear Complexity	0.739918	10/10	Pass

As can be seen, in some key aspects, the obtained sequence does not meet the statistical standards of randomness. This is caused by technical limitations because, at high n values, $n(n+1)$ overflows the variable and starts to acquire a certain pattern, losing the aspect of randomness.

This issue can be prevented by using libraries for working with large numbers (for example, the GNU Multiple Precision Arithmetic Library [58] or bn from OpenSSL [59]) or by optimizing the algorithm itself, such as by changing the series we iterate. To save computational resources, we will focus on the latter option.

Let's analyze the series iteration in the given algorithm. For each iteration, the value of $n(n+1)$ is calculated, which transforms into n^2+n. As can be seen, the value in the denominator of the series increases quadratically, which is the reason for the low speed and loss of randomness characteristics in the later iterations of the algorithm. One can try to simplify this series by removing one of the multipliers and checking the statistical characteristics of the sequence obtained in this case the $n(n+1)$ contains two multipliers: even and odd. The best option would be to leave the odd multiplier because, in the case of an even one, the first bit of the iteration value of the series will always be 0, and when calculating the numerator, the obtained value will be even, which represents a certain pattern of the pseudorandom sequence.

As a result, we get the following improved algorithm:

Input: sequence_length
n: = 1
numerator_buffer = 0
denominator_buffer = 1
sequence[sequence_length]
Repeat sequence_length times
sequence_iteration = n
numerator_buffer = numerator_buffer *
sequence_iteration+denominator_buffer
denominator_buffer *= sequence_iteration
If (numerator_buffer "1) >= denominator_buffer:
sequence[i-th] = 1
numerator_buffer = (numerator_buffer "1" —
denominator_buffer
If numerator_buffer == 0:
denominator_buffer = 1
Otherwise:
sequence[i-th] = 0
numerator_buffer "=1

n+ = 2
Output: sequence[]

After testing the improved algorithm using the NIST statistical test suite, we obtained the following results:

Analyzing the test results, we can conclude that the algorithm generates sequences that meet the statistical standards for pseudorandom sequences. Let's move on to comparing the performance between the basic version and the improved one.

Comparison of the Improved Algorithm with the Basic Version:

Technical Specifications of the Computer:
CPU: AMD Ryzen 5 4500U with Radeon Graphics (6) @ 2.375GHz
OS: Linux
RAM: 16 GB
Sequence Generation Performance:
10,000 bits generation:
Basic: 0m0.003 seconds
Improved: 0m0.002 seconds
1,000,000 bits generation:
Basic: 0m0.105 seconds
Improved: 0m0.081 seconds

100,000,000 bits generation:
Basic: 0m4.401 seconds
Improved: 0m4.300 seconds

Table 2
Test results of an improved pseudorandom sequence generation algorithm based on the calculation of the Taylor series using the NIST statistical test suite

Statistical Test	p-value	Pass Rate	Status
Frequency	0.122325	10/10	Pass
Block Frequency	0.739918	10/10	Pass
Cumulative Sums	0.066882	10/10	Pass
Runs	0.008879	10/10	Pass
Longest Run	0.534146	9/10	Pass
Rank	0.350485	10/10	Pass
FFT	0.534146	10/10	Pass
Non-Overlapping Template	0.739918	10/10	Pass
Overlapping Template	0.066882	10/10	Pass
Universal	0.213309	10/10	Pass
Approximate Entropy	0.350485	9/10	Pass
Random Excursions	–	2/2	Pass
Random Excursions Variant	–	2/2	Pass
Serial	0.739918	10/10	Pass
Linear Complexity	0.534146	10/10	Pass

From these results, we see that the improved algorithm consistently performs faster than the basic version at all tested sequence lengths, demonstrating its efficiency. This shows significant advantages, especially when the algorithm scales to larger data sizes, suggesting that the modifications made to simplify the series calculation contribute to a reduction in computational time while maintaining or enhancing the randomness quality of the sequences.

As can be seen from the figure, at any sequence length, the improved algorithm shows better performance, and considering that it also passes the NIST statistical tests, this indicates its significant advantage over the basic algorithm. This improvement not only enhances efficiency but also ensures that the algorithm maintains robust statistical properties, making it highly effective for applications requiring high-quality pseudorandom sequences.

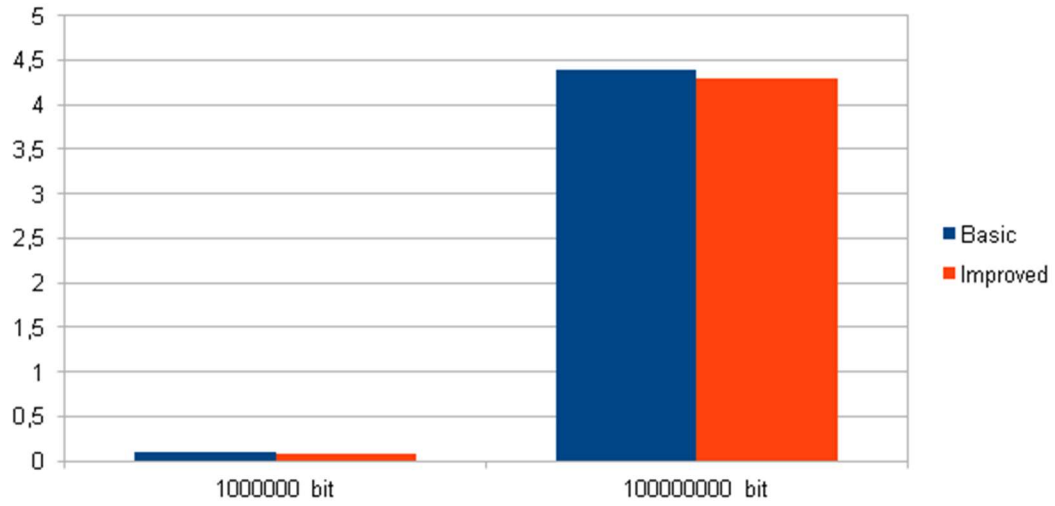

Figure 1: Shows a comparison of the performance between the basic and improved algorithms. The blue color represents the basic version, and the orange color represents the improved version

5. Conclusion

The main conclusions of our research include:

Algorithm Development: A new algorithm based on the Taylor series has been proposed that provides the generation of pseudorandom sequences. This approach is based on the numerical properties of the natural logarithm of number 2 (ln 2), which is mathematically stable and accurate. Using ln 2 to initialize the generator allows achieving a high degree of randomness in the created sequences.

Algorithm Analysis: A detailed analysis of the developed algorithm was conducted, which includes checking its statistical characteristics and testing for compliance with NIST requirements. Testing showed that the algorithm could not initially provide a uniform distribution of pseudorandom numbers, leading to its improvement.

Algorithm Improvement: The basic algorithm has been improved, which provides better performance and improved statistical characteristics of the generated sequences. Optimization of the algorithm allows for significantly reducing the computational complexity, making it effective for use in real-world applications where computation time is a critical parameter.

The results of this research are an important step towards improving the reliability and quality of pseudorandom number generators. The proposed approach may find wide application in various fields such as cryptography, numerical modeling, simulations, and other numerical methods that require high-quality randomness and computational efficiency.

Furthermore, the improved algorithm proposed in this paper can be used to create new generators or to enhance existing solutions, for example through optimization of calculations or application of new generation methods. Future research may focus on expanding the algorithm to other mathematical constants, which may further improve the quality of pseudorandom numbers. It is also possible to create an algorithm based on formula (5) using intervals (for example, as in Hamming matrices) or using other Taylor series for generating new pseudorandom sequences. Using such methods opens new horizons for the development of number theory and computational mathematics, providing powerful tools for solving a wide range of tasks in various fields of science and technology, especially for information protection.

References

[1] E. Alkan, A Comparative Study on Pseudo Random Number Generators in IoT devices, Delft University of Technology, Bachelor Seminar of Computer Science and Engineering (2021).

[2] T. Kietzmann, M. Wählisch, A Guideline on Pseudorandom Number Generation (PRNG) in the IoT, ACM Comput. Surv. 54 (2022) 1–38. doi: 10.1145/3453159.

[3] M. Mandrona, V. Maksymovych, Comparative Analysis of Pseudorandom Bit Sequence Generators, J. Automation Inf. Sci. 49(3) (2017) 78–86. doi: 10.1615/JAutomatInfScien.v49.i3.90.

[4] G. Fishman, Pseudorandom Number Generation, Discrete-Event Simulation, Springer Series in Operations Research, (2001). doi: 10.1007/978-1-4757-3552-9_9.

[5] M. François, D. Defour, P. Berthomé, A Pseudo-Random Bit Generator Based on Three Chaotic Logistic Maps and IEEE 754-2008 Floating-Point Arithmetic, Theory and Applications of Models of Computation, TAMC 2014, LNCS 8402, (2014). doi: 10.1007/978-3-319-06089-7_16.

[6] E. Barker, L. Feldman, G. Witte, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, ITL Bulletin, National Institute of Standards and Technology, Gaithersburg, MD (2015).

[7] L. Shujun, M. Xuanqin, C. Yuanlong, Pseudo-random Bit Generator Based on Couple Chaotic Systems and Its Applications in Stream-Cipher Cryptography, Progress in Cryptology—INDOCRYPT 2001, LNCS 2247 (2001). doi: 10.1007/3-540-45311-3_30.

[8] V. Buriachok, et al., Invasion Detection Model using Two-Stage Criterion of Detection of Network Anomalies, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 2746 (2020) 23–32.

[9] V. Sokolov, P. Skladannyi, H. Hulak, Stability Verification of Self-Organized Wireless Networks with Block Encryption, in: 5th International Workshop on Computer Modeling and Intelligent Systems, vol. 3137 (2022) 227–237.

[10] M. TajDini, et al., Brainwave-based Authentication using Features Fusion, Comput. Secur. 129, no. 103198 (2023) 1–11. doi: 10.1016/j.cose.2023.103198.

[11] Z. B. Hu, et al., Authentication System by Human Brainwaves Using Machine Learning and Artificial Intelligence, in: Advances in Computer Science for Engineering and Education IV (2021) 374–388. doi: 10.1007/978-3-030-80472-5_31.

[12] V. Maksymovych, M. Mandrona, O. Harasymchuk, Dosimetric Detector Hardware Simulation Model Based on Modified Additive Fibonacci Generator, in: Advances in Computer Science for Engineering and Education II, Springer International Publishing, Cham (2020) 162–171.

[13] L. Baldanzi, et al., Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm, Sensors 20(7) (2020). doi: 10.3390/s20071869.

[14] V. Maksymovych, et al., Simulation of Authentication in Information-Processing Electronic Devices Based on Poisson Pulse Sequence Generators, Electronics (Basel) 11 (2022). doi: 10.3390/electronics11132039.

[15] V. Maksymovych, et al., Development of Additive Fibonacci Generators with Improved Characteristics for Cybersecurity Needs, Appl. Sci. (Basel) 12 (2022) 1519. doi: 10.3390/app12031519.

[16] A. Orúe, et al., A Review of Cryptographically Secure PRNGs in Constrained Devices for the IoT, SOCO 2017, ICEUTE 2017, CISIS 2017: International Joint Conference SOCO'17-CISIS'17-ICEUTE'17 649 (2018). doi: 10.1007/978-3-319-67180-2_65.

[17] L. Baldanzi, et al., Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm, Sensors 20(7) (2020) 1869. doi: 10.3390/s20071869.

[18] E. Almaraz Luengo, A Brief and Understandable Guide to Pseudo-Random Number Generators and Specific Models for Security, Statistic Surveys 16 (2022) 137–181.

[19] E. Faure, et al., Method for Generating Pseudorandom Sequence of Permutations Based on Linear Congruential Generator, CMIS-2022: The Fifth International Workshop on Computer Modeling and Intelligent Systems (2022).

[20] D. Lehmer, Mathematical Methods in Large-Scale Computing Units, Second Symposium on Large-Scale Digital Calculating Machinery (1949) 141–146.

[21] R. Katti, R. Kavasseri, V. Sai, Pseudorandom Bit Generation Using Coupled Congruential Generators, IEEE Transactions on Circuits and Systems II: Express Briefs 57(3) (2010) 203–207. doi: 10.1109/TCSII.2010.2041813.

[22] G. Jonatan, et al., Gaussian Pseudo-Random Number Generator using LFSR's Rotation and Split, International Symposium on Electronics and Smart Devices (ISESD) (2021) 1–5. doi: 10.1109/ISESD53023.2021.9501694.

[23] S. Ichikawa, Pseudo-Random Number Generation by Staggered Sampling of LFSR, Eleventh International Symposium on Computing and Networking (2023) 134–140, doi: 10.1109/CANDAR60563.2023.00025.

[24] V. Maksymovych, O. Harasymchuk, M. Shabatura, Modified Generators of Poisson Pulse Sequences Based on Linear Feedback Shift Registers, Advances in Intelligent Systems and Computing, AISC 1247 (2021) 317–326.

[25] C. Nouar, Z. Guennoun, A Pseudo-Random Number Generator Using Double Pendulum, Appl. Math. Inf. Sci. 14 (2020) 977–984. doi: 10.18576/amis/ 140604.

[26] A. Jagannatam, Mersenne Twister—A Pseudo Random Number Generator and its Variants, Technical report, Department of Electrical Computer Engineering, George Mason University (2008).

[27] V. Maksymovych, et al., A New Approach to the Development of Additive Fibonacci Generators Based on Prime Numbers, Electronics (Switzerland) 10(23) (2021) 2912. doi: 10.3390/electronics10232912.

[28] V. Maksymovych, et al., Hardware Modified Additive Fibonacci Generators Using Prime Numbers, Advances in Computer Science for Engineering and Education VI, ICCSEEA 2023, LNDECT 181 (2023). doi: 10.1007/978-3-031-36118-0_44.

[29] S. Bilan, M. Bilan, S. Bilan, Novel pseudo-Random Sequence of Numbers Generator Based on Cellular Automata, Collection Inf. Technol. Secur. 3(1) (2015) 38–50. doi: 10.20535/2411-1031.2015. 3.1.57710.

[30] I. Hanouti, et al., A Lightweight Pseudo-Random Number Generator Based on a Robust Chaotic Map, Fourth International Conference On Intelligent Computing in Data Sciences (ICDS) (2020) 1–6. doi: 10.1109/ICDS50568. 2020.9268715.

[31] Z. Chen, Z. Niu, A. Winterhof, Arithmetic Crosscorrelation of Pseudorandom Binary Sequences of Coprime Periods, IEEE Transactions on Information Theory 68(11) (2022) 7538–7544. doi: 10.1109/TIT.2022.3184176.

[32] A. Zamula, A. Semchenko, Pseudo-Random Number Generators Based on Discrete Logarithm, Technology Audit and Production Reserves 5(1(13)) (2013) 28–31. doi: 10.15587/2312-8372.2013.18390.

[33] R. Gennaro, An Improved Pseudo-Random Generator Based on Discrete Log, Advances in Cryptology — CRYPTO 2000, CRYPTO 2000, LNCS 1880 (2000). doi: 10.1007/3-540-44598-6_29.

[34] C. Lara-Nino, A. Diaz-Perez, M. Morales-Sandoval, Elliptic Curve Lightweight Cryptography: A Survey, IEEE Access 6 (2018) 72514–72550. doi: 10.1109/ACCESS.2018.2881444.

[35] D. Bailey, P. Borwein, S. Plouffe, On The Rapid Computation of Various Polylogarithmic Constants (1997).

[36] J. Napier, Mirifici Logarithmorum Canonis Descriptio (1614).

[37] C. Boyer, A History of Mathematics, Wiley (1991).

[38] E. Maor, e: The Story of a Number, Princeton University Press (1994).

[39] T. Apostol, Mathematical Analysis, Addison-Wesley (1974).

[40] C. Lanczos, Applied Analysis, Dover Publications (1988).

[41] W. Press, et al., Numerical Recipes: The Art of Scientific Computing, Cambridge University Press (2007).

[42] D. Sweeney, On the Computation of Euler's Constant (1962).

[43] J. Steward, Calculus, Brooks Cole (2002).

[44] D. Bailey, S. Plouffe, Recognizing Numerical Constants (1995).

[45] X. Gourdon, P. Sebah, N-th Digit Computation (2003).

[46] X. Gourdon, P. Sebah, The Logarithmic Constant: log 2 (2004).

[47] William Echols. URL: https://williamechols.com/#ln 2

[48] Number World. URL: http://www.numberworld.org/y-cruncher/records.htm

[49] Number World. URL: http://www.numberworld.org/y-cruncher/

[50] Number World. URL: http://www.numberworld.org/y-cruncher/#Features

[51] Number World. URL: http://www.numberworld.org/y-cruncher/#Records

[52] NIST. URL: https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software
[53] NIST SP 800-22, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.
[54] O. Garasimchuk, et al., A Study of the Characteristics of the Fibonacci Modified Additive Generator with a Delay, J. Autom. Inf. Sci. 48(11) (2016) 76–82. doi: 10.1615/JAutomatInfScien.v48.i11.70.
[55] V. Maksymovych, et al., Investigating the Statistical Characteristics of Poisson Pulse Sequences Generators Constructed in Different Ways, J. Autom. Inf. Sci. 49(10) (2017) 11–19.
[56] V. Maksymovych, O. Harasymchuk, I. Opirskyy, The Designing and Research of Generators of Poisson Pulse Sequences on Base of Fibonacci Modified Additive Generator, International Conference on Theory and Applications of Fuzzy Systems and Soft Computing, ICCSEEA 2018: Advances in Intelligent Systems and Computing 754 (2019) 43–53. doi: 10.1007/978-3-319-91008-6_5.
[57] B. Susanti, J. Jimmy, M. Ardyani, Evaluation with NIST Statistical Test on Pseudorandom Number Generators based on DMP-80 and DMP-128, 5th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI) (2022) 166–171. doi: 10.1109/ISRITI56927.2022.10053041.
[58] GNU Multiple Precision Arithmetic Library. URL: https://gmplib.org/
[59] OpenSSL Documentation for the bn Library. URL: https://www.openssl.org/docs/man1.0.2/man3/bn.html